Proceedings of the 7TH BANFF KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Banff Conference Centre
Banff, Alberta, Canada
October 11-16, 1992

In Cooperation with the
American Association for
Artificial Intelligence

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

Proceedings of the
7TH BANFF KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Banff Conference Centre, Banff, Alberta, Canada
October 11-16, 1992

In Cooperation with the
American Association for Artificial Intelligence

Workshop Co-Chairs:
Brian R. Gaines, University of Calgary
Mark A. Musen, Stanford University
John H. Boose, Boeing Computer Services

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.

7TH BANFF KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Sponsored by the American Association for Artificial Intelligence
Banff, Canada, October 11-16, 1992

CONTENTS

Program v
List of Participants x

Technical Papers (listed alphabetically by first author)

On the Specification of Knowledge Model Components 1-1
Manfred Aben, University of Amsterdam

Knowledge Acquisition Techniques for Group Decision Support 2-1
John H. Boose, Jeffrey M. Bradshaw, Joseph L. Koszarek, David B. Shema, Boeing Computer Services

Sharable Ontologies as a Basis for Communication and Collaboration in Conceptual Modeling 3-1
Jeffrey M. Bradshaw, Peter D. Holm, John H. Boose, Boeing Computer Services

Knowledge Acquisition with a Knowledge-Intensive Machine Learning System 4-1
Clifford A. Brunk, Michael J. Pazzani, University of California, Irvine

An Operational KADS Modelling Language and Tool Support for its Application 5-1
Simone Bürsner (University of Heidelberg), Thomas Wetter (IBM Germany)

Inferential Model and Inferential Modelling Technique: A Systematic Technique for Knowledge Analysis 6-1
Christine W. Chan, University of Regina

Graph-Grammar Assistance for Modeling of Decisions 7-1
John W. Egar, Angel R. Puerta, Mark A. Musen, Stanford University

Task Modeling with Reusable Problem-Solving Methods 8-1
Henrik Eriksson, Yuval Shahar, Samson W. Tu, Angel R. Puerta, Mark A. Musen, Stanford University

An Analysis of Languages which Operationalize and Formalize KADS Models of Expertise 9-1
Dieter Fensel, Rudi Studer, University of Karlsruhe

KWrite: Incorporating Active Knowledge Bases in Multi-Media Documents 10-1
Brian Gaines & Mildred L. G. Shaw, University of Calgary

Impact of a GDSS for Multiple Expert Knowledge Acquistion 11-1
Martha Grabowski, Le Moyne College and Rensselaer Polytechnic Institute

A Translation Approach to Portable Ontology Specifications 12-1
Thomas R. Gruber, Stanford University

Object Classification Model (OCM): A Graph-Based Technique for Knowledge Acquisition and Representation 13-1
Kunihiko Higa (Georgia Institute of Technology), Yihwa Irene Liou (University of Baltimore)

The Active Glossary: Taking Integration Seriously 14-1
Georg Klinker, David Marques, John McDermott, Therese Mersereau, Linda Stinson, Digital Equipment Corporation

Integrating Techniques for Conceptual Modeling 15-1
Timothy C. Lethbridge, Doug Skuce, University of Ottawa

Using the Operational Modeling Language OMOS to Represent KADS Conceptual Models 16-1
Marc Linster, GMD

Increasingly Automated Procedure Acquisition in Dynamic Systems 17-1
Nathalie Mathé, Smadar T. Kedar, NASA Ames Research Center

Using Causality as a Generic Knowledge Representation, or Why and How Centralised Knowledge Servers Can Use Causality 18-1
Tim Menzies, Ashesh Mahidadia, Paul Compton, University of New South Wales

Adaptation of a task specific problem solving method to the features of an application 19-1
Klas Orsvärn, Swedish Institute of Computer Science

Further Research in Task-Specific Integration Architectures for Heterogeneous Problem-Solvers 20-1
William Punch III, Michigan State University

The value of Design Knowledge 21-1
Yoram Reich, Carnegie Mellon University

Knowledge Acquisition in the Small 22-1
Issues in Building Knowledge-Acquisition Tools from Pieces
Jay T. Runkel, William P. Birmingham, University of Michigan

Cases, Models and Integrated Knowledge Acquisition to Formalize Operators in Manufacturing 23-1
Gabriele Schmidt (German Research Center), Monika Zickwolff (Technical University Darmstadt)

RepGrid-Net: Combining Conceptual Modeling with Electronic Mail to Provide Group Decision Support 24-1
Mildred L. G. Shaw & Brian Gaines, University of Calgary

Using Script-Based Objects For Knowledge Description in GSE 25-1
Glenn G. Shephard, San Jose State University

Modeling in EMA 26-1
Susan Spirgi, Dieter Wenger, Swiss Bank Corporation

The Knowledge-Based Methodology-Application EMA 27-1
Susan Spirgi, Dieter Wenger, Swiss Bank Corporation

A Study on Task Descriptions Usefulness for Task Knowledge Acquisition 28-1
Yuri Tijerino*, David Marques**, Riichiro Mizoguchi*, Georg Klinker**, Tadahiro Kitahashi*, John McDermott** (*Osaka University) (**Digital Equipment Corporation)

Design Issues for Design Knowledge Elicitation 29-1
Andrew Tunnicliffe (De Montfort University), Stephen Scrivener (Loughborough University of Technology)

Generalised Directive Models 30-1
Gertjan van Heijst, Peter Terpstra, Bob Wielinga (University of Amsterdam), Nigel Shadbolt (Nottingham University)

The CommonKADS Framework for Knowledge Modelling 31-1
Bob Wielinga*, Walter Van de Velde**, Guus Schreiber*, Hans Akkermans*** (* University of Amsterdam), (** Free University of Brussels), (*** Netherlands Energy Research Foundation ECN)

PROGRAM

7TH BANFF KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

In Cooperation with the American Association for Artificial Intelligence
Banff, Alberta, Canada, October 11-16, 1992

Co-Chairs
Brian R. Gaines, University of Calgary
Mark Musen, Stanford University
John Boose, Boeing Computer Services

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

DAILY SCHEDULE

Workshop sessions will be held in the Bougeau Lounge on Floor 6 of the Lloyd Hall building. Continental breakfasts will be available before each session in the workshop area at 7:30. Sessions will start each day at 9:00. There will be coffee and juice breaks each morning and afternoon, and lunch will be served from 12:20 - 1:20, also in the workshop area. Hot breakfasts will be available in the cafeteria in the Donald Cameron building (main residential building). There will be **group dinners** Monday, Tuesday, and Thursday evenings in a **dining room reserved** for KAW'91 in the Donald Cameron building.

This meeting is a workshop, not a conference, and the program is targeted on promoting discussion on strategic and technical issues of importance to the KA community. Session speakers should use their 10 minute slot to address the issues of the session in relation to their own work, and not attempt to give a detailed research paper. Most of the time is scheduled for group discussion—from past workshops, it is never enough!

SCHEDULE OF EVENTS

Sunday, October 11

4:00 - 6:00	Registration; muffins, coffee & tea
6:30	Dinner (in KAW dining room)
7:30 pm	Reception: Welcome & announcements, nuts, punch

Monday, October 12

Note: the first two days are now planned as a plenary mini-workshop with the theme of knowledge modeling.

Overview

9:00 - 9:20	Workshop Welcome and Introduction
9:20 - 9:40	An overview of the Workshop on Problem-Solving Methods —Mark A. Musen, Stanford University

Plenary Session—Issues of Knowledge Modeling (20 min. presentations plus 20 min. discussion)

9:40 - 10:20	A Translation Approach to Portable Ontology Specifications —Thomas R. Gruber, Stanford University
10:20 - 10:40	Break
10:40 - 11:20	Knowledge Acquisition in the Small: Issues in Building Knowledge-Acquisition Tools from Pieces —William P. Birmingham, University of Michigan
11:20 - 12:00	Towards the Group Memory: GDSS & KA —Jay Nunamaker, University of Arizona
12:00 - 1:00	Lunch
1:00 - 1:40	The CommonKADS Framework for Knowledge Modelling —Guus Schreiber, University of Amsterdam

Plenary Session—KADS Operationalization (10 min. presentations plus 60 min. discussion)

1:40 - 3:10	An Analysis of Languages which Operationalize and Formalize KADS Models of Expertise —Dieter Fensel, University of Karlsruhe An Operational KADS Modelling Language and Tool Support for its Application —Simone Bürsner, University of Heidelberg On the Specification of Knowledge Model Components —Manfred Aben, University of Amsterdam
3:10 - 3:30	Break

Plenary Session—Knowledge Modeling (10 min. presentations plus 60 min. discussion)

3:30 - 5:00	Modeling in EMA —Susan Spirgi, Swiss Bank Corporation The Knowledge-Based Methodology-Application EMA —Dieter Wenger, Swiss Bank Corporation Generalised Directive Models —Gertjan van Heijst, University of Amsterdam
5:00 - 6:00	Poster preparation time—for those who haven't put their posters up yet—pins, yarn, all kinds of goodies available
6:30	Dinner (in KAW dining room)

Poster Reception

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

Tuesday, October 13

Plenary Session—Task Modeling (10 min. presentations plus 60 min. discussion)

9:00 - 10:30 A Study on Task Descriptions Usefulness for Task Knowledge Acquisition
—Yuri Tijerino, Osaka University
Task Modeling with Reusable Problem-Solving Methods
—Henrik Eriksson, Stanford University
Adaptation of a task specific problem solving method to the features of an application
—Klas Orsvärn, Swedish Institute of Computer Science

10:30 - 10:50 Break

Plenary Session—Knowledge Integration (10 min. presentations plus 60 min. discussion)

10:50 - 12:20 The Active Glossary: Taking Integration Seriously
—Georg Klinker, Digital Equipment Corporation
Integrating Techniques for Conceptual Modeling
—Timothy C. Lethbridge, University of Ottawa
Cases, Models and Integrated Knowledge Acquisition to Formalize Operators in Manufacturing
—Gabriele Schmidt, German Research Center

12:20 - 1:20 Lunch

Plenary Session—Ontologies (10 min. presentations plus 60 min. discussion)

1:20 - 2:50 Using Causality as a Generic Knowledge Representation, or Why and How Centralised Knowledge Servers Can Use Causality
—Tim Menzies, University of New South Wales
Graph-Grammar Assistance for Modeling of Decisions
—John W. Egar, Stanford University
Sharable Ontologies as a Basis for Communication and Collaboration in Conceptual modeling
—Jeffrey M. Bradshaw, Boeing Computer Services

2:50 - 3:10 Break

Plenary Session—Group Knowledge Modeling (10 min. presentations plus 60 min. discussion)

3:10 - 4:40 Knowledge Acquisition Techniques for Group Decision Support
—John Boose, Boeing Computer Services
Impact of a GDSS for Multiple Expert Knowledge Acquistion
—Martha Grabowski, Le Moyne College and Rensselaer Polytechnic Institute
RepGrid-Net: Combining Conceptual Modeling with Electronic Mail to Provide Group Decision Support
—Mildred L. G. Shaw, University of Calgary

6:30 Dinner (in KAW dining room)

Wednesday, October 14

Note: Wednesday and Thursday mornings are multi-tracked as decided at KAW'91. One track has been allocated to papers addressing a variety of themes other than knowledge modeling, and this will have use of the main meeting rooms. Other tracks will be arranged at the meeting and will use the smaller lounges.

Special Interest Group Tracks

9:00 - 10:30 Meetings (to be arranged)

10:30 - 10:50 Break

10:50 - 12:20 Meetings (to be arranged)

Paper Track—Design Knowledge/Machine Learning (10 min. presentations plus 50 min. discussion)

9:00 - 10:30 Design Issues for Design Knowledge Elicitation
—Andrew Tunnicliffe, De Montfort University
The value of Design Knowledge
—Yoram Reich, Carnegie Mellon University
Knowledge Acquisition with a Knowledge-Intensive Machine Learning System
—Clifford A. Brunk, University of California, Irvine
Increasingly Automated Procedure Acquisition in Dynamic Systems
—Nathalie Mathé, NASA Ames Research Center

10:30 - 10:50 Break

Paper Track—Knowledge Elicitation (10 min. presentations plus 50 min. discussion)

10:50 - 12:20 Using Script-Based Objects For Knowledge Description in GSE
—Glenn G. Shephard, San Jose State University
Object Classification Model (OCM): A Graph-Based Technique for Knowledge Acquisition and Representation
—Yihwa Irene Liou, University of Baltimore
Inferential Model and Inferential Modelling Technique: A Systematic Technique for Knowledge Analysis
—Christine W. Chan, University of Regina
Further Research in Task-Specific Integration Architectures for Heterogeneous Problem-Solvers
—William Punch III, Michigan State University

12:20 - 1:20 Lunch—in cafeteria, or boxed lunches may be ordered (on Monday) to take out

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

6:30 Dinner (in KAW dining room)

Thursday, October 15

Note: Thursday afternoon is allocated to plenary report-backs from the different tracks, and open discussion on themes arising at the meeting.

Special Interest Group Tracks

9:00 - 10:30 Meetings (to be arranged)

10:30 - 10:50 Break

10:50 - 12:20 Meetings (to be arranged)

12:20 - 1:20 Lunch

Plenary Session—Reports from Special Interest Groups

1:20 - 2:50 Reports from various groups and general discussion

2:50 - 3:10 Break

3:10 - 4:40 Reports from various groups and general discussion

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

Friday, October 16

Note: Friday morning is allocated to a re-evaluation of the need for, and role of, future meetings, planning activities during the coming year, including EKAW93.

Plenary Session—Review and Plans for Future

9:00 - 9:30 Project Sisyphus—Activities to date and Plans for Future
—Marc Linster et al

9:30 - 10:30 Plans for 1993 Activities
—Guy Boy, Mark Musen, Tom Gruber et al

10:30 - 10:50 Break

10:50 - 12:20 Where do we go next as a community?
—All

12:20 - 1:00 Farewell Lunch

LIST OF PARTICIPANTS

7th BANFF
KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Banff, Canada, October 11-16, 1992

Manfred Aben
Social Science Informatics
University of Amsterdam
Roetersstraat 15, NL-1018
Amsterdam, The Netherlands

Hans Akkermans
Free University of Brussels
AI Lab
Lentestraat 50-54, B-1050 Brussels
Belgium

William P. Birmingham
EE & Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122

Guy A. Boy
Institut Européen de l'Ingénierie et des Sciences Cognitives
Prologue 1, B. P. 27-25
3132 Labège Cedex
France

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

Clifford A. Brunk
Information & Comp.Science
University of California
Irvine, CA 92717

Simone Bürsner
University of Heidelberg
Medical Biometry and Informatics
IM Neuenheimer Feld 400
W-6900 Heidelberg, FRG

Christine W. Chan
Dept. of Computer Science
University of Regina
Regina, Saskatchewan S4S 0A2

John W. Egar
Medical Informatics
Stanford University
Stanford, CA 94305-5479

Henrik Eriksson
Medical Comp. Science Group
Knowledge Systems Laboratory
Stanford University, MSOB,
Stanford, CA 94305-5479

Dieter Fensel
Instit.Angewandte Informatik
Universitat Karlsruhe
PO Box 6980, 7500 Karlsruhe
Germany

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

Martha Grabowski
Decision Sciences & Engineering
Rensselaer Polytechnic Institute
Troy, New York 12180-3590

Thomas R. Gruber
Knowledge Systems Laboratory
Stanford University
701 Welch Rd., Bldg. C
Palo Alto, CA 94025

Markus Khauff
Instit. für Informatic Research
ABT: Kognitionswissenschaft
Friedrichstr 50
D-7000 Freiburg Germany

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

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

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

Marc Linster
AI Research Division
GMD
PO Box 1316, D-5205
Sankt Augustin 1
Germany

Yihwa Irene Liou
R G Merrick School of Business
University of Baltimore
1420 North Charles Stret,
Baltimore, MD 21201-5779

Nathalie Mathé
NASA Ames Res. Center
Mail Stop 269-2
Moffett Field, CA 94035

Tim Menzies
Artificial Intelligence Laboratory
University of New South Wales
PO Box 1, Kensington
New South Wales, Australia 2033

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

Klas Orsvärn
S. I. C. S.
PO Box 1263, 164 28 Kista
Sweden

William Punch
Dept. of Computer Science
Michigan State University
A-714 Wells Hall
East Lansing, MI 48824-1027

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

Yoram Reich
Engineering Design Res. Center
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213

Jay T. Runkel
EE & Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122

Gabriele Schmidt
German Research Center for AI
PO Box 2080
Erwin Schrodinger-Str
D-6750 Kaiserslautern, Germany

Guus Schreiber
University of Amsterdam
Dept. of S. W. I.
Roetersstraat 15, NL-1018 WM
Amsterdan The Netherlands

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

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

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

Susan Spirgi
Swiss Bank Corporation
10L-WBS, Hochstrasse 16
CH-4022 Basel, Switzerland

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

Pedro Szekely
USC/ISI
4676 Admiralty Way
Marina del Rey
CA 90292, USA

Yuri Adrian Tijerino
Institute of Scientific and
Industrial Research
Osaka University
8-1 Mihogaoka, Ibaraki-shi
Osaka 562, Japan

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

Andrew Tunnicliffe
Dept. of Computer Science
James Went Building
De Montfort University
Leicester LE1 9BH England

Gertjan van Heijst
Dept. of Psy., S. W. I.
University of Amsterdam
Roeterstraat 15, NL - 1018 WB
Amsterdam, The Netherlands

Dieter Wenger
Swiss Bank Corporation
10L-WBS, Hochstrasse 16
CH-4022 Basel, Switzerland

On the Specification of Knowledge Model Components*

Manfred Aben
Department of Social Science Informatics
University of Amsterdam
Roetersstraat 15, NL-1018 WB Amsterdam, The Netherlands
e-mail: manfred@swi.psy.uva.nl

Abstract

In this paper some of the problems with using predefined building blocks to specify knowledge level models of problem solving are outlined, in particular in the context of the KADS methodology. The definition of the basic building blocks in KADS, the *primitive inferences*, or *knowledge sources*, often seems to be inadequate to aid the knowledge engineer in constructing an abstract model of problem solving. We argue that the informal, verbal way in which the building blocks are defined is the cause for this problem, and propose to formalise them to make their semantics clear and to assess the consequences of various modeling decisions. We do not claim that there is no need for informal knowledge level models; we suggest that formalisation can be helpful during the construction and validation of knowledge level models and of the knowledge based system

1 Introduction

A number of knowledge engineering methodologies (e.g. the Generic Task approach [Chandrasekaran, 1986], the Method–to–Task approach [Musen *et al.*, 1988], the Role–limiting Method approach [McDermott, 1988; Klinker *et al.*, 1991], the Components of Expertise approach [Steels, 1990]) are based on the notion of reusable components, which are used to build models of problem solving. A central issue is how these reusable components are defined.

In this paper some of the problems with the current definition of reusable components will be discussed, and a possible solution to them will be outlined in the context

*The research reported here was carried out in the course of the KADS-II project, partially funded by the ESPRIT Programme of the CEC as project number P5248. The partners in this project are Cap Gemini Innovation (F), Cap Gemini Logic (S), the Netherlands Energy Research Foundation ECN (NL), Entel (ESP), IBM France (F), Lloyd's Register of Shipping (UK), the Swedish Institute of Computer Science (S), Siemens AG (D), Touche Ross Management Consultants (UK), University of Amsterdam (NL) and the Free University of Brussels (B).

of the KADS methodology for the construction of knowledge based systems. The basic reusable components in this methodology are the *inferences* or *knowledge sources*. The problems can roughly be divided into two categories: firstly the ambiguity of the names and descriptions of the inferences, and secondly the fact that the domain ontology on which the inferences are defined is too general for specific tasks. As a solution to the first problem, we propose to formalise inferences in first order logic, making all their assumptions and properties explicit, clarifying relations between inferences and revealing new inferences. To deal with the second problem, we show how inferences based on other, task–oriented ontologies, can be described and related to the inferences based on the KADS domain ontology.

This paper is structured as follows: the next section discusses the concept of reusable components in various approaches to knowledge acquisition and outlines the main problems with these components. Section 3.1 give a concise overview of those aspects of KADS which are relevant to this paper, and outlines the main problems with primitive inferences addressed in this paper. Section 4 presents a solution to the problems presented in section 2.1, by means of formalisation of the set of primitive inferences. The choices that are to be made when formalising primitive inferences are outlined, the formal language is presented, and as an illustration an example inference is formalised. Section 5 describes how task–specific inferences can be described in the formalisation, and how they relate to the ontology–specific inferences. Finally, section 6 discusses the results and relations to other research in the field.

2 Reusable Components In Knowledge Acquisition

The Generic Task Approach The approach taken in [Chandrasekaran, 1986; Bylander & Chandrasekaran, 1988] is to construct a model of a particular problem solving task, by combining a set of predefined generic tasks. These generic tasks contain not only a decomposition of the task that is modeled, but also an inference mechanism to solve the tasks. The selection of the set of generic tasks that can solve the problem at hand is guided by analysing features of the task. It is assumed that there is a finite set of generic tasks, which may need some tuning to suit the domain of application. This tuning is a underspecified enterprise, and seems to be viewed mainly as a programming exercise. The reusability of the generic tasks is limited, due to their rather large grainsize. Recently the generic task approach seems to be moving towards a set of smaller generic tasks with extended reusability.

The Components of Expertise Approach The approach outlined in [Vanwelkenhuysen & Rademakers, 1990; Steels, 1990] is based on the decomposition of a task on the basis of characteristics of its input and output, on the availability of domain models and on the basis of pragmatical or environmental constraints. A *Problem solv-*

ing methods is either a *decomposition method* or an *action method.* A decomposition method decomposes a task into smaller subtasks, specifies control over these subtasks, and determines the knowledge requirements of the subtasks. An *action method*, solves the task *directly.* Selection of either these methods can be done by the knowledge engineer analysing the task to be modeled, or dynamically by the target knowledge based system. This approach assumes a library of tasks, methods and domain models, and connections between these. For instance, some task can be solved by various methods. Which method is selected to solve the task is determined dynamically; on what basis one method is to be preferred over another is not entirely clear. It is assumed that methods can be refined to suit particular tasks.

The Role–limiting Method Approach In [Klinker *et al.*, 1991; Marques *et al.*, 1991] an integrated system is described, based on the role–limiting method approach, that aids non–programmers in the construction of expert systems. The system consists of three subsystems: Spark, which aids the user in identifying and refining an *activity model*, and which realises the identification of *methods* to perform the activities. The Burn tool is used to acquire the *domain knowledge* that is required by the methods. The FireFighter system identifies weaknesses or errors in the constructed system and suggest repairs.

The approach relies on a library of reusable programming constructs or *mechanisms.* Mechanisms can be configured into *methods* of various grainsize. A mechanism is characterised by the information that it uses and the information that is produces. Besides a library of mechanisms, the workbench contains a library of predefined *activity* models. Each pre–defined task–activity is associated to pre–defined methods. This association also connects the task–specific terminology to the task–inspecific mechanisms.

The PROTÉGÉ–II Approach In [Tu *et al.*, 1991] a programme that generates method–specific knowledge acquisition tools is described. The system is based on three notions:

As in the role–limiting method approach, this approach identifies *methods* and *mechanisms.* Methods can be decomposed into methods, or they are mechanisms. Methods and mechanisms are characterised by their input an output, in terms of a *Global Data Model* (or: GDM). A GDM contains a vocabulary in which to express the roles that domain knowledge can play in a range of tasks. Methods and mechanisms are specifications of the transformation from the input to the output, in terms of transformation rules on the knowledge level. When something is called a mechanism or a method is called a *design decision*, although it would probably be better labeled a *modeling decision.* The distinction between these mechanisms and the ones that are used in the role–limiting method approach is that these mechanisms contain no computational assumptions.

The KADS Approach In the KADS [Wielinga *et al.*, 1992; Breuker & Wielinga, 1985; Schreiber *et al.*, 1989] methodology KBS development is seen as a modeling activity. The specification and design of a knowledge based system is viewed as constructing and refining models of problem solving behaviour. In this process, the conceptual model plays a crucial role. The conceptual model is a knowledge level description of the problem solving behaviour of some agent (human or artefact). This knowledge level description is divided into four knowledge categories or *layers*:

- The *domain layer*, describing all static, axiomatic domain knowledge in terms of a basic use–independent ontology based on the epistomological primitives proposed by [Brachman & Schmolze, 1985].
- The *inference layer*, describing the *roles* that domain knowledge can play and the *inferences* that can be made in terms of these roles, in an *inference structure*, which is essentially a input/output dependency graph.
- The *task layer* specifies control (called *task structures*) over these inferences, and how a task–decomposition relates to the inference structure.
- The *strategy layer* specifies the goals that are to be reached in order to solve some problem, and generates or selects task structures that can solve a particular problem.

The inferences are characterised by the ontological characterisation of their input and output, and by the domain relations that are used to specify the transformation from input to output. As the ontology in which the domain is expressed is fixed, and contains a small set of primitives, KADS assumes that a small set of inferences can be described in terms of these ontological primitives. The inferences are specified on the knowledge level, hence they make no computational assumptions. How an inference is computationally realised is specified in the *design model.*

2.1 Problems with the Building Blocks in the Various Approaches

This paper shares the view with [Musen *et al.*, 1988] that *guiding* the configuration of building blocks is very important, since that is what modeling with reusable components is all about!

Several problems emerge from all of the approaches mentioned in the previous section.

The reusable components are underspecified The various building blocks are described rather informally, and often ambiguously. The terminology used for describing the inputs and output determines how specific the description of the inference that operates on it can be described.

The rather vague term *feature* or *property* is often used as a indexing mechanism to libraries of reusable components, but it is hardly ever described clearly what these features are, and how they are used to select the components.

Within the various methodologies the building blocks are not all equally generic or reusable. However, the methodologies do not provide any guidance to *compare* the building blocks, or to determine how they interrelate. The decomposition of building blocks into smaller building blocks in itself reveals the relation between them, but it is not clear *why* some decomposition is valid, or *why* one decomposition is preferred over another decomposition. Also the interrelations between various building blocks, which could be the basis of the selection of appropriate building block, is often underspecified.

As the description of the building blocks, and the relations between them are underspecified (or at least ambiguous) it is difficult to asses the consequences of selecting one component rather than another. Selecting a building block, for instance, might impose a specific structure on the domain knowledge, or might restrict the available building blocks later on.

Ontologies determine reusability of the building blocks The various methodologies use different ontologies to describe the knowledge that is used by their building blocks. The ontologies range from very task–specific (Generic Tasks, Component of Expertise), to very use–independent like the KADS ontology. There is a clear trade–off between the descriptiveness of the ontology, in the sense that it coincides with the terminology of the task that is modeled, and the reusability of the ontology, in the sense that it can be used to model various tasks. The reusability of the building blocks is clearly related to their grainsize, which is in turn related to the descriptiveness of the ontology.

Most approaches distinguish between a base–ontology, and a task specific ontology, that can be mapped onto the base–ontology. Hence a set of generic inferences can be specified in terms of the base–ontology. In the role–limiting method approach, the components of expertise approach and the generic task approach, the mapping between the inferences that operate on the base–ontology and those that operate on the task specific ontologies is predefined. The PROTEGE–II system, and to a larger extend, the KADS methodology consider it part of the modeling process to make this mapping. These mappings are never one–to–one and the support that the various methodologies provide to make these mappings is limited.

This paper addresses both problems by looking at the various aspects of the building blocks in a *formal* way.

3 Scoping The Problem

Rather than looking at all of the aforementioned methodologies superficially, this paper will focus the way the problems described in the last section reveal themselves in KADS.

3.1 Concise overview of KADS

This section briefly describes the KADS approach to knowledge level modeling. It is not claimed nor intended to be complete in its description, but merely reiterates some of the ideas which are fundamental to this paper. The reader is referred to [Breuker & Wielinga, 1985; Schreiber *et al.*, 1989] and in particular to [Wielinga *et al.*, 1992] for a more elaborate description of the methodology.

In the KADS methodology KBS development is seen as a modeling activity. The specification and design of a knowledge based system is viewed as constructing and refining models of problem solving behaviour. In this process, the conceptual model plays a crucial role. The conceptual model is a knowledge level description of the problem solving behaviour of some agent (human or artefact). This knowledge level description is divided into four knowledge categories.

The first category is called the domain knowledge. This layer contains all static, axiomatic domain knowledge. The knowledge in this category is formulated in a use–independent way, which supports its reusability. The formulation of this category is in terms of a KL–ONE [Brachman & Schmolze, 1985] like ontology.

The second category is the inference knowledge. This type of knowledge defines canonical inferences, elementary steps in the reasoning process, which derives new information from existing knowledge. The inference steps are characterised by their input and output and additional domain knowledge. The input and output are descriptions of the *role* some domain knowledge element plays in the inference process. The additional domain knowledge is used to derive the output from the input(s). An inference structure describes all sanctioned inference steps and roles, and their interrelations.

The third knowledge category describes the task knowledge. This type of knowledge specifies how inference steps are combined and applied to solve a specific problem. A task can be decomposed into simpler tasks, and it specifies control over the subtasks. The leaves of this task decomposition refer to canonical inferences at the inference level.

The fourth category is the strategic knowledge. This type of knowledge specifies the goals that are to be reached in order to solve some problem. Strategic knowledge therefore generates or selects task structures that can solve a particular problem.

The four categories are in fact *layers* in the conceptual model, where each layer refers to the layer directly below it. The strategic layer specifies ar selects task structures at the task layer, the leaves of the task structures refer to inferences at the inference layer, and the inferences and knowledge roles refer to domain knowledge at the domain layer.

3.2 The Basic Ontology

The ontology used in KADS to describe the domain knowledge is based on the epistomological primitives proposed by [Brachman & Schmolze, 1985]. This ontology consists of:

- *concepts*
- *instances* of concepts
- *sets* of instances
- *attributes* defined on instances and concepts
- *relations* between concepts or between instances, such as the *is–a* and *part–of* relations
- *structures* consisting of concepts or instances and their relations.

In [Wielinga *et al.*, 1992] an extension to this ontology is suggested:

- *expressions* over attributes and values
- *value–relations*, i.e. relations between attribute expressions

The *conceptualisation* of a domain, i.e. carving up the domain along the ontological primitives, is not a trivial task. It is often hard to decide which ontological primitive is suitable to model some part of the domain. Often various conceptualisations of the same knowledge are conceivable.

3.3 A Typology of Inferences

Given the KADS ontology, we can describe (primitive) operations that can be performed on the ontological primitives. In [Breuker *et al.*, 1987] a typology of primitive inferences is presented, which is based on the ontological categories of the input and output knowledge roles on which they operate. The proposed ontology and the proposed typology of canonical inferences have a number of serious flaws. Bredeweg [Bredeweg, 1992] discerns four types of flaws[1]:

1. *The ontology is not complete.* In particular, Bredeweg observes that the ontology lacks an explicit definition for sets of (other) ontological primitives, although the typology makes use of sets.

2. *The typology is incomplete.* Since the ontology lacks a definition for sets, an extra set operation **merge** is introduced. It is unclear why other set operations such as intersection and element removal, are not mentioned in [Bredeweg, 1992]. Furthermore, not all possible combinations of input and output roles (in terms of ontological characterisation) are described. One could even question if a finite set of inference types exists. Obviously, if we *define* primitive inferences as operations on ontological primitives, and we have a finite set of ontological primitives, the number of n–ary relations that can be defined over these primitives is infinite if n is not fixed, but not all of these are "useful" and "meaningful" inferences.

[1]The flaws recognised in [Bredeweg, 1992] are in italics.

3. *The labels of inference steps are confusing.* The names of the inferences (e.g. **abstract**, **specify**) often have a specific meaning in either artificial intelligence, or in the colloquial sense, which does not coincide with their use in KADS.

4. *Some domain– or task-specific ontologies do not map naturally onto the* KL–ONE *ontology.* Although it seems possible to map various ontologies onto the KL–ONE ontology, the resulting inferences are intuitively not very appealing. Bredeweg argues the need for inferences such as *generate hypotheses.*

An additional weakness of the proposed typology of inferences is that they model the operations on ontological categories of domain knowledge, but they do not model operations on the "roles" that domain knowledge plays in the inference process. This is an advantage in the sense that it gives a small set of primitives which seems adequate to model many problem types and areas, hence supporting knowledge reusability. The generality of the ontology, however, makes it often difficult to model an inference as one from the typology. This is because one tends to specify the inferences in as operations on the *roles* that the domain knowledge plays in the inferencing, and not in terms of their *ontological structure.* As KADS provides only a set of inferences in which are specified in terms of the base–ontology, the resulting inferences are often not intuitive.

For example, the notion of *hypothesis* or *observation* can be modeled as concepts or as value expressions, but having to specify inferences that deal with hypotheses or observations, in terms of concepts and attributes is often difficult. One would rather have a more task–oriented ontology in which to specify inferences, and specify the mapping between this ontology and the KADS ontology. In this way, we could look at the task–oriented ontology as a set of *predefined conceptualisations* of knowledge roles in terms of the underlying KADS ontology. To use programming languages as a metaphor: the task–oriented ontology is a set of *macros* specified on top of the KADS base–ontology.

Another problem with the various proposed inferences is, that they are not all at the same level of complexity; they are not all equally primitive, if you want[2]. For instance, an inference step like **transform** which transforms one structure into another, can hardly be called primitive and lacks a clear description. This is one of the reasons why this inference is often used as a "catch–all".

In [Breuker *et al.*, 1987] it is stated that inferences are characterised not only by their input– and output roles, but also by the (type of) domain relations they refer to:

> [...] *in connection with the domain* [a knowledge source] *can be further specified by its reference to a particular type of relation at the domain level, and a particular method that exploits this type of relation to produce the particular type of inference.*
> [Breuker *et al.*, 1987; page 32]

[2]This is the main reason why the term *canonical inference* has been coined.

In the proposed typology, the inferences are hardly ever described along this dimension. It is our view that relations are indeed a distinguishing factor in the description of typologies, and provide a bridge towards a more task–oriented, although still reusable set of inferences.

3.4 An Example Inference: Abstract

To illustrate the problems with the current definition of primitive inferences, let us look at **abstract**:

> *Formally, in abstraction the relation between the input concept and the output concept is that the latter contains less attributes than the former: 'irrelevant' attributes are abstracted. What is relevant and what is irrelevant depends on a* point of view.
> [Breuker *et al.*, 1987; page 37]

> *Abstract is defined as removing an attribute from a concept. The abstract knowledge source makes a concept less specific by abstracting away an irrelevant detail (see also specify). Abstract may also work on an instance.*
> [Bredeweg, 1992; page 50]

These definitions give a quite clear, albeit rather restricted notion of abstraction. However, things get less clear in [Breuker *et al.*, 1987]:

> *If only the point of view is emphasised —and not the reduction of attributes— abstraction is a very general term for relating a concept with another concept; almost as a synonym for any type of inference. One may* **abstract** *that a patient has fever if his temperature is higher than 36°C ('qualitative abstraction', [Clancey, 1985]). Points of view may be prewired in the knowledge base (domain layer). For instance, many structures of static relations (*is_a, consists_of*) have fixed points of view, and enable abstract inferences.*
> [Breuker *et al.*, 1987; page 37]

It is unclear how such a qualitative abstraction can be viewed as removing an attribute. In terms of the domain ontology, one could argue that such an abstraction is in fact the derivation of an attribute (*fever*) from another attribute (*temperature*).

This type of qualitative abstraction involves the derivation of a value of some attribute of some concept or instance, from other attributes of that concept or instance. In this sense, abstraction results in *more* attributes rather than *less*. Hence, abstraction defined as such coincides with KADS's **specify** inference, rather than **abstract**, it's exact opposite! Even if we replace the concrete attributes with the abstract attribute,

we have a combination of **abstract** and **specify** [3]. This is not to say that the naming of **abstract** is unappropriate for the type of inference that it defines. Indeed, we are likely to call the removal of attributes of some concept, abstraction. But one would like to have primitives modeling the above type of abstraction as well, and be able to distinguish the various types of abstraction.

The confusion resulting from the definition is apparent if we look at the proceedings of the second KADS user meeting [Bauer & Karbach, 1992]. Out of nine papers that use the **abstract** inference, only one author uses it in correspondence with the original definition. Most of the other papers use an **abstract** which models qualitative or definitional abstraction, and rely on the availability of an abstraction hierarchy at the domain layer. There are also authors who use the **abstract** inference syntactically incorrect, i.e. with an incorrect number of inputs and outputs.

With respect to the KADS domain ontology and the inferences defined upon it, a number of authors verbalised their discontent:

> *Using the suggested knowledge sources helps to build an inference structure, but their strict and exact typifying — which in a programming language is quite helpful for writing exact algorithms — is much more narrowing than giving flexibility as it is required in building a somewhat intuitive conceptual model.*
> [Löckenhoff & Reminger, ; page 16]

> *On one hand it is not clear what the exact characteristics or semantics of the existing knowledge sources are and on the other hand this given typology seems to be quite incomplete.*
> [Poeck & Gappa, ; page 10]

> *The criterion* [...] *used* [...] *to classify knowledge sources is a pure syntactical one.* [...] *it would be a better approach to classify knowledge sources semantically by their purpose.*
> [Köppen *et al.*, ; page 15]

> *The typology might come in handy if it is associated with a library of templates for each of the classes and if the semantics of each of the classes can be stated more precisely.*
> [Landes *et al.*, ; page 8]

4 Formalising The Primitive Inferences

In a number of papers (a.o. [Karbach *et al.*, 1991; Wetter & Schmidt, 1991]) the need to formalise knowledge level models has been put forward. In particular, in the context of

[3] And not simply **abstract** as [Bredeweg, 1992; page 53] suggests.

KADS, a number of papers discuss ways to formalise inference structures ([Akkermans *et al.*, 1989; Linster, 1992; Wetter, 1992; Kowalczyk & Treur, 1990; Angele *et al.*, 1991; van Harmelen & Balder, 1992]). Our approach is in line with [van Harmelen & Balder, 1992] in the sense that it is not the intention to *replace* the informal knowledge level model, but rather complement it, by providing support in constructing the knowledge level model. Also, we do not aim at a computationally attractive formalisation or operationalisation of the inferences, in contrast with most of the work on formalisation of knowledge level models ([Linster, 1992; Wetter, 1992; Kowalczyk & Treur, 1990; Angele *et al.*, 1991]). Our goal is to resolve the problems with the KADS typology of primitive inferences as outlined above.

4.1 Why Logic?

The case for logic to disambiguate informal, verbal specifications has been made before ([Moore, 1982; Hayes, 1977]). To summarise, the advantages of using a logic formalisation to describe knowledge level models are:

- The semantics of the language is fixed, and there is a uniform interpretation of the language, i.e. *it reduces ambiguity.*
- It becomes possible to *derive and formulate properties of the knowledge level constructs*, such as conditions which must hold to make the inference applicable, conditions that hold after an inference has been made, computational properties, e.g. does the inference reduce the search space, *etc.* Hence, *the implications of modeling decisions become more clear* opening the way for better support during knowledge level modeling.
- Having a clear description of the knowledge level constructs makes it possible to *compare and relate knowledge level constructs.* In this way we can derive whether one inference subsumes the other, whether one inference is the inverse of another, and so forth.

Obviously, the cost of using a formal language to specify a knowledge level model, is that it forces one to be very precise and detailed. From a knowledge engineering point of view, formalisation should take place only when the informal model becomes unclear, if some modeling decisions are not apparent, or if the informal model is accepted, and a bridge is to be made towards the *design model* of the knowledge based system that is to be constructed.

4.2 Formalisation Choices

We now have a language[4] in which to specify the inferences. However, we can formalise the inferences in various ways. We could try to encode everything we can say about the inference into one complex formula, but this is often not very useful, since it obfuscates the essence of the inference. Instead, we will distinguish various aspect of primitive inferences, and formalise them separately:

Parameters Rather than making very specific formalisations, one can choose to parametrise the inferences. Many of the primitive inferences are parametrised over predicates, e.g. $\leq_{subs}$, which reads "subsumed by". The actual definition of the subsumption relation is then not explicit in the formalisation. One can either see the definition of such a predicate as input to the inference, or as distinguishing various instantiations of the more general inference.

Invariants In general, it is not easy to indicate that an output concept is in fact a modified input concept. In the formalisations below, a *new* output concept is created. This means we have to take care that the only changes with respect to the input concept are the ones prescribed by the primitive inference. This would mean that we have to specify *invariants*. To avoid the frame–problem we assume that inferences have no side–effects, i.e. the only changes that are induced by an inference are the ones prescribed by its definition.

Pre– and Postconditions Instead of formalising inferences in great detail, we have chosen to separate the various aspects of an inference. Parametrising, as discussed earlier is one way of separating these aspects from the "core" of the inference. Other aspects are the conditions under which an inference may be applied. For instance, conditions about the form of the input knowledge roles. Still other aspects are those that apply to the output of the inference: which conditions should hold after application of the inference. Typically, when an inference step is parametrised over some relation, the definition and aspects of that relation are usually part of the pre– and postconditions. For instance, if we relate two instances over a non–transitive relation, we must ensure that this non–transitivity is respected by the inference step.

Strengthening and Weakening We may choose to strengthen certain inferences, either by making their definition more specific, or by making the pre– or post–conditions more specific. Weakening is the obvious inverse of this, and we may also choose to weaken some aspects and to strengthen others. This gives us variants of

[4]The language used in the rest of this paper is first order predicate logic. In fact, the formalisations which will be discussed later fit in the $(ML)^2$approach [van Harmelen & Balder, 1992]. In this paper we focus on the *knowledge source predicates* and will not specify all the details from proper $(ML)^2$

the inference. In fact, strengthening and weakening is equivalent to traversing some semantic typology of inferences top–down or bottom–up respectively.

4.3 Ontology and Notation

We use an order sorted first order logic (cf. [van Harmelen & Balder, 1992]). The ontological primitives used in the definition of primitive inference steps are listed below, with their syntactical representation used in this paper. The characters used to distinguish various ontological categories are written in lowercase. The uppercase characters refer to variables which range over the ontological category represented by their lowercase equivalent. Variables are assumed to be universally quantified, unless they are explicitly quantified.

instance An instance is formalised as a constant. In the following instances will be named ι, possibly subscribed or with accents.

- **instance set** A set S is a comma separated list of instances, embraced by curly brackets: $\{\iota_1, \ldots, \iota_n\}$.

concept A concept is a unary predicate, of which the argument is an instance, or a variable that ranges over a set of instances. Concepts will be denoted as $c(I)$, or, when the instances on which the concept is defined are not relevant, abbreviated to c. Note that the quantification of the instances over which the concept ranges is not specified. E.g. does $c(I)$ mean: $\forall I\ c(I)$ or $\exists I\ c(I)$? By default we will assume the universal quantification.

attribute Attributes are defined as functions over instances: f : instance $\longrightarrow$ value. A value v is the result of a function application on an instance: $f(\iota)$.

- **value expression** A value expression is an n–ary *function* over the values of attributes. The result of a value expression is itself a value, and hence may be used as an argument to another value expression. To put it differently, the most primitive value expression is the attribute itself. Expressions will be denoted as: $\epsilon(f_1(\iota), \ldots, f_n(\iota))$.
- **value relations** Value relations specify how the value of an attribute is determined by values of other attributes. This relation will be denoted as an n–ary *predicate* defined over value expressions. A typical example of a value relation is a *value restriction.* Value relations will be denoted as: $\varrho(\epsilon_1, \ldots, \epsilon_n)$. Value relations can also be higher order, i.e. they may be defined over other value relations: $\varrho(\varrho_1, \ldots, \varrho_n)$.
- **attribute/value pair** Attribute value pairs are specific value relations, the relation in this case is equality. Hence they will be denoted as equations of function applications and terms: $f(\iota) = v$. To specify that some attribute

is defined on some instance, without knowing, or caring about, the value of that attribute, we write: $f(\iota)\downarrow$, which is a shorthand for: $\exists\, V f(\iota) = V$. To specify that some attribute is not defined on some instance we write $f(\iota)\uparrow$, a shorthand for $\neg\exists\, V f(\iota) = V$. Note that attributes are defined on instances only. In order to specify that an attribute is defined over all instances of some concept, we can write $\forall I\ c(I) \rightarrow f(I)\downarrow$, and we introduce a shorthand notation $f(c)\downarrow$. Similarly we define $f(c)\uparrow$ and $f(c) = v$.

relation instance A relation will be denoted as an n–ary predicate defined over instances, where $n \geq 1$: $r(\iota_1, \ldots, \iota_n)$ is called a *relation instance*. To indicate that all instances of some concept are related to all instances of another concept, we may write: $\forall I_1, I_2\ c_1(I_1) \wedge\ c_2(I_2) \rightarrow r(I_1, I_2)$ or as a shorthand $r(c_1, c_2)$. Note that we may chose to define the relation between concepts differently, e.g. by assuming other quantifications of the instances of the concepts involved. In that case the quantifications need be explicitly mentioned.

structure A structure $\mathcal{S}$ is defined and denoted by a set of relation instances: $\mathcal{S} = \{r_1(\iota_1, \ldots, \iota_n), \ldots, r_k(\iota'_1, \ldots, \iota'_m)\}$. The definition allows for structures having multiple relations, but does not allow for "isolated" instances, i.e. instances that are not related to any other instance.

list A list L is an ordered set of instances without duplicates, embraced by angle brackets: $\langle \iota_1, \ldots, \iota_n \rangle$. Since the list is ordered, there is a relation between all successive instances in the list. Hence, a list is a specific type of structure, where all relation instances are binary relations over successive instances in the list: $\langle \iota_1, \ldots, \iota_n \rangle \equiv \{r(\iota_1, \iota_2), \ldots, r(\iota_{n-1}, \iota_n)\}$ for some ordering r.

graph A graph $\mathcal{G}$ is a denoted by two sets, $\mathcal{V}$ and $\mathcal{E}$ containing the *vertices* or *nodes* and the *edges* respectively, and a relation $\mathcal{R}$. Hence, $\mathcal{V}$ is a set of instances and $\mathcal{E}$ is a set of relation instances, where each relation instance is denoted as a list of two instances. Note that a graph represents only one binary relation. Depending on this relation the graph may be *directed*, *transitive*, *acyclic* etc. Notation:
$\mathcal{G} = \langle \mathcal{R}, \mathcal{V}, \mathcal{E} \rangle$ where $\mathcal{V} = \{\iota_1, \ldots, \iota_n\}$ and $\mathcal{E} = \{\langle \iota_1, \iota'_1 \rangle, \ldots, \langle \iota_n, \iota'_n \rangle\}$
A graph, too is a specific type of structure:
$\langle \mathcal{R}, \{\iota_1, \ldots, \iota_n\}, \{\langle \iota_1, \iota'_1 \rangle, \ldots, \langle \iota_n, \iota'_n \rangle\} \equiv \{\mathcal{R}(\iota_1, \iota'_1), \ldots, \mathcal{R}(\iota_n, \iota'_n)\}$

We have chosen to place the quantifiers such, that the formulae can be easily read. Also, to avoid numerous brackets, the formulas are indented, which should intuitively make the scope of the quantifiers and operators clear.

4.4 Example formalisation

It is important to note that the formal definitions of the inferences below are in fact "higher order": the formulae range over the ontological categories of the domain,

where these categories are formalised as first order formulae themselves! This higher–orderness is not made explicit in the formalisation of inferences. For instance, variables seem to range over function–symbols. To be formally correct, the ontological categories need be *named* (see [van Harmelen, 1991]) at the inference layer, i.e. formulae on the domain layer should be referred to as terms at the inference layer. This naming is however not explicit in the formalisations below, as it would seriously affect readability.

$$c, c' : \mathit{concept}; p : \mathit{predicate} \tag{1}$$

$$\mathbf{abstract}(c, p, c') \Leftrightarrow \tag{2}$$

$$F(c)\downarrow \ \wedge\ p(F) \rightarrow F(c')\uparrow \tag{3}$$

The predicate p is the *irrelevance criterion*: for each attribute F, F is irrelevant iff $p(F)$. This formalisation is a literal translation of the original definition. It states that all irrelevant attributes F defined on the input concept c, are no longer defined on the output concept c'. This formalisation is *parametrised* over p, that means the definition of p is not part of the definition of abstract. Typically, the relevance criteria are defined at the domain layer. An *invariant* could be, that all the *relevant* attributes are left intact:

$$F(c)\downarrow \ \wedge\neg p(F) \rightarrow F(c')\downarrow \tag{4}$$

If one wants to enforce that only concepts which have irrelevant attributes are to be abstracted, one could define the *precondition*:

$$\exists F : \mathit{attribute} \tag{5}$$

$$F(c)\downarrow \ \wedge\ p(F) \tag{6}$$

As a *postcondition*, one could derive that the number of attributes does not increase:

$$F(c)\uparrow \rightarrow F(c')\uparrow \tag{7}$$

We can *strengthen* this definition in a number of ways. For instance, by enforcing that there are no irrelevant attributes defined on the abstract concept (i.e. strengthening the postcondition):

$$F(c')\downarrow \rightarrow \neg p(F) \tag{8}$$

We can *weaken* the definition by allowing *attribute–abstraction*:

c, c' : *concept*; $p, <_{abs}$: *predicate* (9)

abstract$(c, p, <_{abs}, c') \Leftrightarrow$ (10)

$F(c)\downarrow \ \wedge \ p(F) \rightarrow$ (11)

$F(c')\uparrow$ (12)

$\vee$ (13)

$\exists F'$: *attribute* (14)

$F'(c')\downarrow \ \wedge \ F <_{abs} F'$ (15)

It is trivial to show that the formalisation of **abstract** in lines (1) through (3) logically imply the formalisation in lines (9) through (15), which shows that the latter is in fact a *weakening* of the former.

This formalisation is also *parametrised* over $<_{abs}$, which is the abstraction relation over attributes. Nothing is said about the *value* of the attributes, by strengthening this definition one could enforce some relation between the values of the concrete attributes and those of the abstract attributes.

5 Task–oriented Inferences

As discussed in section 3.2 there is a need to define inferences in a way that is more natural to model a specific task. This in turn requires that there is a taskspecific ontology of the inputs and outputs of inferences, i.e. an ontology of *knowledge roles.* The problem, however, is that these roles hardly ever have an explicit semantics, they are merely names of knowledge categories. However, if we can fix a set of such names, and establish what "KADS ontological primitives" constitute the knowledge that the names refer to, we can then formalise inferences over these roles, in terms of the "KADS ontology" [5]. This relies on the assumption that knowledge roles can indeed be expressed in terms of the KADS ontology.

One should note that some of the inferences that are used to model specific tasks *are* task and/or domain related, but others *are not.* For instance, the rather syntactic primitive inference **select** is often used alongside more task-oriented inferences.

Let us take the definition of **abstract** above (lines 1 through (3)). It states that abstraction is the removal of irrelevant attributes of some concept or instance. This is often not the way the term "abstraction" is used in many domains. For instance if, in systematic diagnosis, we abstract concrete observations into abstract findings, we do not simply remove an attribute. Usually, we rely on the availability of an *abstraction hierarchy* to perform abstraction. The abstraction hierarchy could be formalised as a set of abstraction rules, each of which is an attribute relation, stating how the value of the abstract attribute is defined by the values of the concrete attributes.

[5] In practice, one will often first define "interesting" inferences, deduce from them what roles are needed, and link these roles to the ontological categories.

$c, c' : concept; \varrho_{abstr} : value_relation$ (16)

abstract$(c, \varrho_{abstr}, c') \Leftrightarrow$ (17)

$\varrho_1(F(c)) \ \wedge \ \varrho_{abstr}(\varrho_1(F(c)), \varrho_2(F'(c))) \rightarrow \varrho_2(F'(c'))$ (18)

This definition states that for all attributes F on c on which some value relation ϱ_1 holds, and for all abstraction relations that specify how that value relation can be abstracted to some value relation ϱ_2 on some abstract attribute F', the value relation ϱ_2 is enforced on the output concept. Note that both ϱ_1 and ϱ_2 are universally quantified.

This caters for abstraction rules such as:

$$\varrho_{abstr}(temperature(patient) = 39, fever(patient) = true) \quad (19)$$

$$\varrho_{abstr}(temperature(patient) > 37.5, fever(patient) = true) \quad (20)$$

$$\varrho_{abstr}(temperature(patient) < 37.5, fever(patient) = false) \quad (21)$$

It is not too difficult to show that this formalisation is a generalisation of the definition given in lines (1) through lines (3)[6].

We might *parametrise* **abstract** with a *set* of abstraction relations and the *attribute* which we want to abstract from:

$c, c' : concept, f : attribute, S_{\varrho_{abstr}} : value_relation_set$ (22)

abstract$(c, f, S_{\varrho_{abstr}}, c') \Leftrightarrow$ (23)

$\varrho_1(f(c)) \wedge \ \varrho_{abstr}(\varrho_1(f(c)), \varrho_2(f'(c))) \in S_{\varrho_{abstr}} \rightarrow \varrho_2(f'(c'))$ (24)

The abstract attribute may depend on many concrete attributes, and we may want to say only which *abstract* attribute we are interested in:

$c, c' : concept, n : integer; f' : attribute, \varrho_{abstr} : value_relation$ (25)

abstract$(c, f', \varrho_{abstr}, c') \Leftrightarrow$ (26)

$\forall_{1 \leq i \leq n} \ F_i(c) \downarrow \ \wedge \ \varrho_{abstr}(\varrho_1(F_1(c), \ldots, F_n(c)), \varrho_2(f'(c))) \rightarrow \varrho_2(f'(c'))$ (27)

If we use more meaningful names, the inference becomes even more readable:

patient,patient' : *concept*; (28)

observation,finding : *attribute*; (29)

abstract_to : *value_relation* (30)

abstract(patient, observation, abstract_to, patient') $\Leftrightarrow$ (31)

ϱ_1(observation(patient)) (32)

$\wedge$ (33)

abstract_to(ϱ_1(observation(patient)), ϱ_2(finding(patient))) $\rightarrow$ (34)

ϱ_2(finding(patient')) (35)

[6]Take $\varrho_1(F(c)) \equiv (F(c) \downarrow \ \wedge p(F))$ and $\varrho_2(F'(c)) \equiv (F' = F \wedge \ F(c) \uparrow)$.

In systematic diagnosis ([Breuker *et al.*, 1987; page 52]) hypotheses are often interrelated over a generality relation: one hypothesis is more general than another. This generality relation can sometimes be given as a hierarchical structure, e.g. a subsumption relation between hypotheses, which can be formalised quite similarly to **abstract** above. In this case the inference would be named **generalise** rather than **abstract**. This exemplifies the fact that the name of inference, is in practice more determined by the type of domain relations that are manipulated by the inference, than the "ontological manipulations" it performs.

6 Discussion

In this paper some of the problems are outlined with building blocks for knowledge level models of problem solving, in particular with KADS' canonical inferences. We proposed a framework in which to formalise the various aspects of inferences, so they may be more easily compared and consequences of modeling decisions become more obvious. In this way, the formalisation can aid the construction of knowledge level models, and also, bridge the gap between knowledge level model and design model.

Before formalising inferences, we first need to establish a set of such inferences. There is probably no fixed or even finite set of inferences. Hence it should be one of the results of this exercise to come up with a *strategy* for formalising inferences. The reader should therefore consider the set of formalised inferences[7] as an extended example of how to go about formalising inferences. In any case, to be able to restrict the number of inference steps possible, we need to restrict the domain ontology, i.e. the ontological categories on which the inferences operate. We have shown that defining the inferences in terms of the ontological categories, and as operations on the ontological categories, when the ontology is very use–independent, makes the resulting inferences often unintuitive and inexpressive for modeling specific tasks. It is proposed that we instead be able to define inferences on the basis of more task–oriented ontologies, which themselves can be mapped on the domain–, use–independent ontology. These task–oriented ontologies are still assumed to range over a large set of domains and tasks.

Some inferences are more specific than others, in terms of what they *do*. For instance, selection of an instance from a set can be specialised by requiring that the instance selected has some unique property. Using this type of dependencies between inferences could serve as an aid for selecting and gradually refining the appropriate inference for a given task. As it seems the case that all of the inferences described in [Breuker *et al.*, 1987] can be differentiated further, it is better to regard these inferences as inference *schemes* or *types*. It is a predominant view in KADS, that in order to aid knowledge engineers in using the KADS methodology, they should be provided with a library of modeling components. We would argue that such a library will hardly ever

[7] In this paper only one formalised inference has been shown, **abstract**. However the complete set of inferences as proposed in [Breuker *et al.*, 1987] has been formalised, and some additional inferences have been identified. Please contact the author for details.

contain "ready–to–use" components, but rather a set of adaptable components. One of the ways to support the knowledge engineer in adapting the selected components, would be the formalisation as presented in this paper. Also, by formalising the inference schemes, specific modeling decisions will become more informed.

Formalising inferences does not in itself resolve the question: "When are inferences to be labeled *primitive* ?" Primitiveness of inferences depends on a.o. the task that is modeled, and on the level of detail the knowledge engineer is interested in. However, as we want to use the formalisation approach to help build knowledge level models, we do not want to formalise inferences in such detail, that specific computational decisions are enforced. To put it differently, refining the formalisations must stop (in this case) when we are modeling design decisions. When we want to use the formalisations as a bridge towards design, we need to clearly distinguish which formal modeling decisions need be reflected in the knowledge level model, and which should not.

Acknowledgements

I would like to thank the following people for many useful comments and extensive discussions on earlier versions of this paper: Bart Benus, Bert Bredeweg, Joost Breuker, Nienke den Haan, Frank van Harmelen, Gertjan van Heijst, Guus Schreiber, Peter Terpstra, Bob Wielinga and the anonymous referees.

References

Akkermans, J. M., Wielinga, B. J., Schreiber, A. T., & Balder, J. R. (1989). Towards a formal specification of knowledge models. Technical Report ECN-89-006, ECN, Petten, The Netherlands.

Angele, J., Fensel, D., Landes, D., & Studer, R. (1991). KARL: An executable language for the conceptual model. In *Proceedings of the 6th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff*, pages 1.1–20, Canada. SRDG Publications, University of Calgary.

Bauer, C. & Karbach, W., editors (1992). *Proceedings Second KADS User Meeting*, ZFE BT SE 21, Otto-Hahn Ring 6, D-8000 Munich 83. Siemens AG.

Brachman, R. J. & Schmolze, J. G. (1985). An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9:171–216.

Bredeweg, B. (1992). *Expertise in qualitative prediction of behaviour*. PhD thesis, University of Amsterdam, Amsterdam.

Breuker, J. A. & Wielinga, B. J. (1985). KADS: Structured knowledge acquisition for expert systems. In *Proc. of the Fifth International Workshop on Expert Systems and their Applications; Avignon, France.*

Breuker, J. A., Wielinga, B. J., van Someren, M., de Hoog, R., Schreiber, A. T., de Greef, P., Bredeweg, B., Wielemaker, J., Billault, J. P., Davoodi, M., & Hayward, S. A. (1987). Model Driven Knowledge Acquisition: Interpretation Models. ESPRIT Project P1098 Deliverable D1 (task A1), University of Amsterdam and STL Ltd.

Bylander, T. & Chandrasekaran, B. (1988). Generic tasks in knowledge-based reasoning: The right level of abstraction for knowledge acquisition. In Gaines, B. & Boose, J., editors, *Knowledge Acquisition for Knowledge Based Systems*, volume 1, pages 65–77. Academic Press, London.

Chandrasekaran, B. (1986). Generic tasks in knowledge-based reasoning: High level building blocks for expert system design. *IEEE Expert*, 1.

Clancey, W. J. (1985). Acquiring, representing and evaluating a competence model of diagnostic strategy. In Chi, Glaser, & Far, editors, *Contributions to the Nature of Expertise.*

Hayes, P. (1977). In defence of logic. In *Proceedings of IJCAI–77*, pages 559–556. International Joint Conference on Artificial Intelligence.

Karbach, W., Voß, A., Schukey, R., & Drouwen, U. (1991). Model-K: Prototyping at the knowledge level. In *Proceedings Expert Systems–91, Avignon, France*, pages 501–512.

Klinker, G., Bhola, C., Dallemagne, G., Marques, D., & McDermott, J. (1991). Usable and reusable programming constructs. *Knowledge Acquisition*, 3:117–136.

Köppen, R., Fensel, D., & Geidel, J. Modeling the Selection of Scheduling Algorithms with KARL. in: [Bauer & Karbach, 1992].

Kowalczyk, W. & Treur, J. (1990). On the use of a formalized generic task model in knowledge acquisition. In Wielinga, B., Boose, J., Gaines, B., Schreiber, G., & van Someren, M., editors, *Current trends in knowledge acquisition*, pages 198–221, Amsterdam. IOS Press.

Landes, D., Hackenberg, D., & Schweier, T. An Inference Structure for a Configuration Problem. in: [Bauer & Karbach, 1992].

Linster, M. (1992). Linking modeling to make sense and modeling to implement systems in an operational environment. In Wetter, T., Althoff, K.-D., Boose, J., & Gaines, B., editors, *Current developments in knowledge acquisition: EKAW92*, volume 509 of *Lecture Notes in AI.* Springer Verlag.

Löckenhoff, C. & Reminger, B. An Inference Structure for Technology Asessment. in: [Bauer & Karbach, 1992].

Marques, D., Dallemagne, G., Klinker, G., McDermott, J., & Tung, D. (1991). Easy programming: Empowering people to build their own applications. Technical report, Digital Equipment Corporation. submitted to the IEEE Expert.

McDermott, J. (1988). Preliminary steps towards a taxonomy of problem-solving methods. In Marcus, S., editor, *Automating Knowledge Acquisition for Expert Systems*, pages 225–255. Kluwer Academic Publishers, The Netherlands.

Moore, R. (1982). The role of logic in knowledge representation and common-sense reasoning. In *National Conference on Artificial Intelligence*, pages 428–433, Pittsburgh, PA. American Association for Artificial Intelligence.

Musen, M. A., Fagan, L. M., Combs, D. M., & Shortliffe, E. H. (1988). Use of a domain model to drive an interactive knowledge editing tool. In Boose, J. & Gaines, B., editors, *Knowledge-Based Systems, Volume 2: Knowledge Acquisition Tools for Expert Systems*, pages 257–273, London. Academic Press.

Poeck, K. & Gappa, U. An Interpretation Model for Heuristic Classification using the Hypothesize–and–Test Strategy. in: [Bauer & Karbach, 1992].

Schreiber, A. T., Bredeweg, B., de Greef, P., Terpstra, P., Wielinga, B. J., Brunet, E., Simonin, N., & Wallyn, A. (1989). A KADS approach to KBS design. ESPRIT Project 1098, deliverable B6 UvA-B6-PR–010, University of Amsterdam & Cap Sogeti Innovation. Available from: University of Amsterdam, Social Science Informatics, Roetersstraat 15, 1018 WB, The Netherlands.

Steels, L. (1990). Components of expertise. *AI Magazine*. Also as: AI Memo 88-16, AI Lab, Free University of Brussels.

Tu, S., Shahar, Y., Dawes, J., Winkles, J., Puerta, A., & Musen, M. (1991). A problem–solving model for episodic skeletal–plan refinement. Report KSL–91-22, Knowledge Systems Laboratory, Medical Computer Science, Stanford University, Stanford (Ca).

van Harmelen, F. (1991). Definable naming relations in meta-level systems. ESPRIT Basic Research Action P3178 REFLECT, Technical Report RFL/UvA/I.4/7, University of Amsterdam. Accepted at META'92.

van Harmelen, F. & Balder, J. R. (1992). $(ML)^2$: a formal language for KADS models of expertise. *Knowledge Acquisition*, 4(1). Special issue: 'The KADS approach to knowledge engineering'.

Vanwelkenhuysen, J. & Rademakers, P. (1990). Mapping knowledge-level analysis onto a computational framework. In Aiello, L., editor, *Proceedings ECAI–90, Stockholm*, pages 681–686, London. Pitman.

Wetter, T. (1992). An executable language for the KADS conceptual and interpretation models. Thesis Draft.

Wetter, T. & Schmidt, W. (1991). Formalization of KADS interpretation models. In Steels, L. & Smith, B., editors, *AISB91, Artificial Intelligence and Simulation of Behaviour*. Springer Verlag.

Wielinga, B. J., Schreiber, A. T., & Breuker, J. A. (1992). KADS: A modelling approach to knowledge engineering. *Knowledge Acquisition*, 4(1). Special issue 'The KADS approach to knowledge engineering'.

Knowledge Acquisition Techniques for Group Decision Support

John H. Boose, Jeffrey M. Bradshaw, Joseph L. Koszarek, David B. Shema

Boeing Computer Science Technology Organization
Boeing Computer Services, 7L-64, PO Box 24346, Seattle, WA, 98124, USA

john@atc.boeing.com, jbrad@atc.boeing.com, jlk@atc.boeing.com,
dshema@atc.boeing.com

Abstract

Existing group decision support systems used in meeting rooms can help teams reach decisions quickly and efficiently. However, the decision models used by these systems are inadequate for many types of problems.

This paper describes our laboratory's experience with knowledge acquisition systems and decision support tools. Our studies lead us to develop a comprehensive decision model for group decision support systems. This decision model combines current brainstorming-oriented methods, structured text argumentation (using the gIBIS model), repertory grids, possibility tables (morphological charts), and influence diagrams from decision analysis. Each component addresses weaknesses in current group decision support systems. We are assembling these group decision support components together into a group decision workbench (Figure 1).

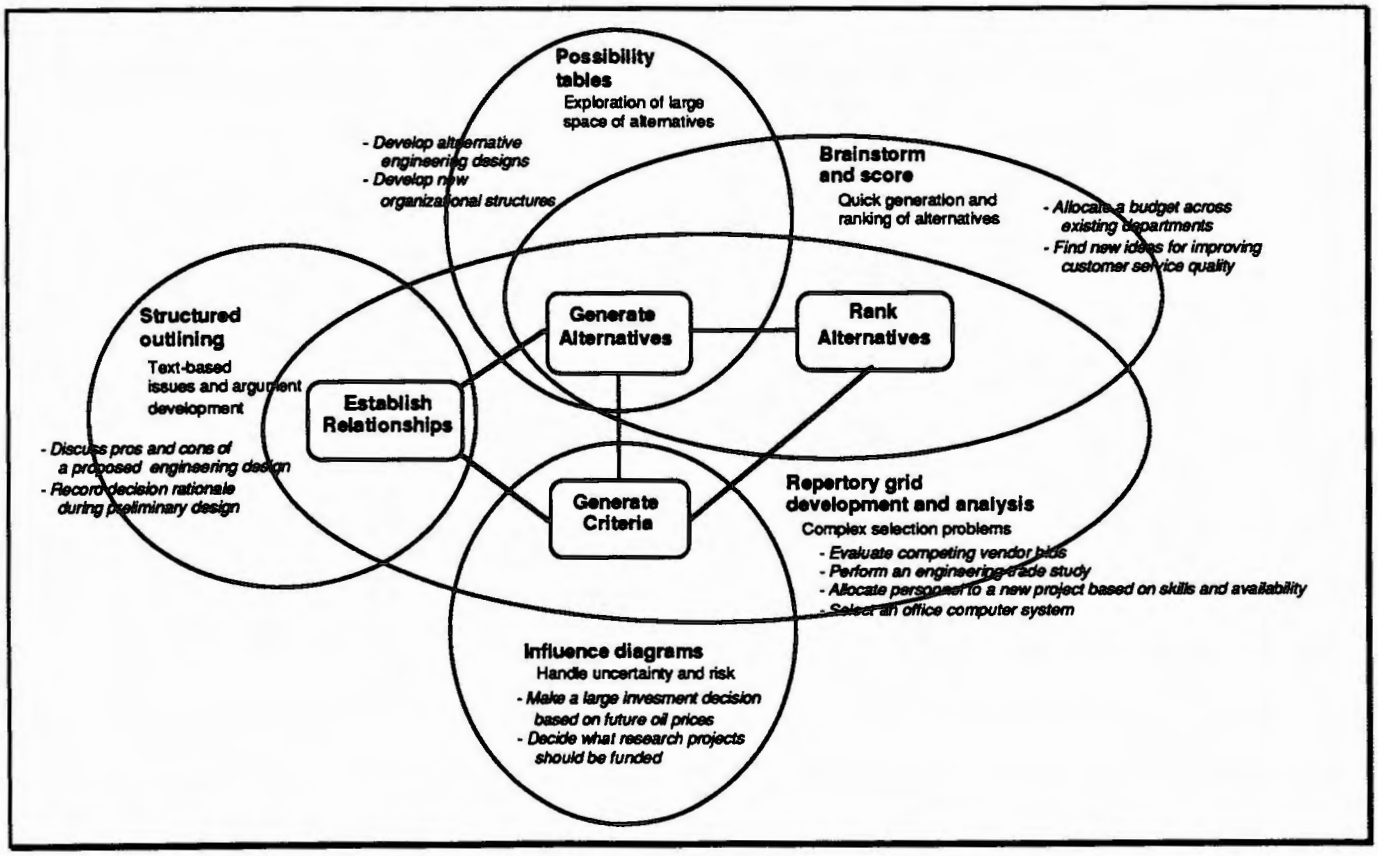


Figure 1. Our decision support model combines components of current group decision support systems and knowledge acquisition methods. We describe the model below.

1.0 Enhancing Group Productivity with Decision Support

To better solve complex problems and foster interdisciplinary work, The Boeing Company has made major organizational changes. For example, the design and fabrication of a commercial airplane requires hundreds of disciplines to cooperatively work together to satisfy customer expectations. These activities require group decisions which integrate sources of information such as customer requirements, technological advancements, and manufacturing methods. Decision alternatives must be evaluated against assumptions, objectives, and constraints. For the next generation airplane, this will require the development of new collaborative information support environments.

To promote the development of a collaborative support environment, we are conducting experiments combining group decision support systems and knowledge acquisition techniques.

1.1 Advantages of Group Decision Support Systems

The need for meeting electronic support is clear. In most organizations people spend an average of 40% of their time in meetings (Table 1).

It is easy to stray from the agenda.
Less assertive individuals may have little opportunity to offer important ideas.
Many meetings end with no sense of closure or accomplishment.
The group must partition available time among members.
Members can't contribute comments as they occur. They may forget or suppress them later in the meeting as they seem less original, relevant, or important.
Fewer comments are made because members concentrate on remembering comments (rather than thinking of new ones) until they can contribute them.
Members must listen to others speak and cannot pause to think and generate new comments.
Members lack focus on communication, missing or forgetting the contributions of others.
Members are reluctant to criticize the comments of others owing to politeness or fear of reprisals.
Fear of negative evaluations cause members to withhold ideas and comments.
Members rely on others to accomplish goals, owing to cognitive loafing, the need to compete for air time, or because they perceive their input to be unneeded.
Discussion moves along one train of thought without deviating. Members don't make comments that are not directly related to the current discussion.
Non-task discussion reduces task performance, although some socializing is usual necessary for effective functioning.
Some group members exercise undue influence or monopolize the group's time in an unproductive way.
Members receive information faster than it can be processed.
Often a strategy is missing for reaching a decision.
Most of the meeting is undocumented.
Scheduling meetings is difficult.
Travel for meetings is expensive.
Unproductive meetings frustrate teams.

Table 1. Barriers to productivity often frustrate meeting participants (Nunamaker Jr., 1991; Nunamaker Jr., Vogel, Heminger, Martz, Grohowski, and McGoff, 1989).

Group decision support systems (GDDS's) address these problems. Tools such as **GroupSystems** (Nunamaker Jr., Applegate, and Konsynski, 1988; Daniels, Dennis, Hayes, Nunamaker Jr., and Valacich, 1991; Ventana, 1990), **SAMM** (Dickson, 1991; Tan, Wei, and Raman, 1991), **TeamFocus** (Nunamaker Jr., Vogel, Heminger, Martz, Grohowski, and McGoff, 1989), and **VisionQuest** (Collaboration Technologies Company, 1991), use the processes of brainstorming, idea organization, and scoring and ranking. These tools support many types of applicatin problems (Table 2).

Budget / resource allocation
Concurrent engineering
Crisis planning
Development of marketing image
Exploration of business challenges
Idea generation
Joint application design (JAD)
Market planning
Negotiation
Policy development
Requirements gathering and definition
Resolution of sensitive issues
Strategic planning
Systems analysis and design
Team building

Table 2. Group decision support systems enable productivity gains in many application areas (Daniels, Dennis, Hayes, Nunamaker Jr., and Valacich, 1991; Dennis, George, Jessup, Nunamaker Jr., and Vogel, 1988; Nunamaker Jr., Weber, Smith, and Chen, 1988; Sycara and Roboam, 1991).

Studies show that groups using GDSS's can solve more problems with greater efficiency (Kramer and King, 1988). Experiments suggest that computer support can improve the *quality* of the decisions (Gallupe, DeSanctis, and Dickson, 1988; Jarvenpaa, Rao, and Huber, 1988; Nunamaker Jr., Applegate, and Konsynski, 1988). Other benefits of electronic meeting room decision support systems appear in Table 3.

General group process advantages:
• A group as a whole has more information than any one member.
• A member uses information in a way that the original holder did not, because that member has different information or skills.
• Groups are better at catching errors than are the individuals who proposed ideas.
• Working as part of a group may stimulate and encourage individuals to do better.
• Members may learn from and imitate more skilled members to improve performance.
Advantages of electronic support:
• Entries made by individuals are anonymous.
• The meeting focus is on content and task, not personalities.
• Simultaneous entry enables equal participation from all members.
• Entries are parallel and simultaneous. One or two individuals do not dominate a meeting. Everyone generate sand evaluates ideas
• Meetings produce electronic records.
• Everyone generates more data in less time.
• Meeting times shorten.
• Fewer meetings are needed.
• IBM on average has reduced calendar time 90% and people hours 60% for many types of meetings.
• It is often easier to reach agreement on volatile issues since ideas are de-personalized.
• Participants feel greater sense of ownership in outcomes.
• The environment imposes a process structure; the structure helps to focus meetings.
• The environment fosters innovation and creativity.

Table 3. GDSS's can help facilitate group processes (Dennis, George, Jessup, Nunamaker Jr., and Vogel, 1988; Nunamaker, Jr., 1991; Nunamaker Jr., Vogel, Heminger, Martz, Grohowski, and McGoff, 1989; Dennis, Valacich, and Nunamaker Jr., 1991).

1.3 Problems with GDSS Decision Models

There are many areas of potential improvement for current GDSS's (such as the user interface, process modeling support, portability, cost). This paper focuses on their *decision modeling ability* in the context of *same time, same place* meetings. Many of the methods discussed below could also be applied to *any time, any place* meetings.

Sometimes the decision models provided by current systems break down when applied to more complex problems. Examine the following scenario:

> Department heads meet with their manager to help allocate money among projects for the next year:
>
> 1) First electronic brainstorming helps produce a list of important project related issues.
> 2) Each member separately allocates a fixed amount of resources among the projects.
> 3) Members further discuss issues and the system displays the group consensus and spread of opinion.
> 4) The manager makes minor adjustments to the consensus and implements the budget.

This *brainstorm-and-score* decision model can work well when there are few interacting constraints among alternatives and when the space of alternatives is fixed in size. But this simple list prioritization scheme cannot represent the following kinds of decision information:

Complex criteria. What reasons are there for picking Project A over Project B? How important are the reasons? How does the group rate each project on each reason? What aspects of criteria do the members prefer? Do certain reasons only apply to certain projects?

Minimums, maximums, ranges. Project C needs a minimum allocation of $200,000. Project D needs a range of $150,000 to $450,000. Our budget limit is $2,000,000.

Restructuring complex alternatives. Projects E, F, and G seem to have overlapping customer requirements and objectives. Is this desirable? Will it lead to duplicated effort or uncommon solutions? Can we restructure projects and their objectives so that we don't duplicate effort but still meet the all requirements?

Exclusivity. If we do Project H, we shouldn't do I.

Enabling conditions. If the full $450,000 were allocated for J, we could purchase critical resources from ACME Products.

Risk management. We have a make or buy decision. We're not sure ACME will supply a critical resource in time for Project K. If they don't, we'll have to build the resource ourselves, and it will cost an additional $200,000. Should we hold this money in a contingency fund? How long should we wait for ACME to deliver the resource? Project L has a similar problem. Can we share the risk across both projects (how is the year's budget affected if both, neither, or one project will need additional money)? What amount of risk should we tolerate?

Effect of uncertainty on payoffs. Project M's success is less certain than that of Project N but would probably have a bigger pay back if it was successful.

Timing. Project O depends on enabling technology from another department. We shouldn't start the project until it's delivered. How can we use the uncertainty about the delivery date to decide how much to allocate to Project O next year? If it is delivered in March we would spend more money next year than if it were delivered in September, even though the multiple-year cost would remain the same.

Documenting running discussions. The requirements changed since the last budget decisions — what arguments contributed to the selection of one alternative over another? What will be the effect on requirement satisfaction if we use a different alternative?

The brainstorm-and-score model can't handle these types of decision information. For instance, it uses a satisficing method — "Find a way to solve the problem," not, "Make sure you find the best way — or at least a good way — to solve the problem." A more complex model can help when we need assurance that a large portion of the space of alternatives has been considered (for instance, engineering design alternatives for a major system or subsystem) or when there are interacting constraints.

Brainstorm-and-score models lack explicit criteria, or at most include one criterion for scoring, such as "importance" or "value". It is often important to illuminate many of the criteria affecting the decision. Explicit criteria can help point out areas of agreement and disagreement and enable special forms of analysis. We can express preferences for criteria values. We can assess uncertainty about aspects of the decision. We can take into account attitudes toward risk and timing of deliverables. We can measure the value of gathering further information or spending more resources to control aspects of a situation. The model can represent constraints between criteria. The reuse of criteria for future changes and similar problems also helps offset the cost of collecting information about criteria.

Current GDSS's offer powerful environments to help users enumerate ideas during a *divergence* phase of activity. But another problem occurs during the *convergence* phase, when users must reduce and categorize the idea list. This synthesis process can be dissatisfying and painful for group members. Methods discussed below can reduce this frustration by providing an information constraint framework (see the section on possibility tables).

Section 2 discusses the foundations of our knowledge acquisition approach for representing this information in a decision model. In Section 3 we develop a model that incorporates the missing decision elements needed to handle these problems. We use the budget problem and other examples to illustrate features of the model.

2.0 A Knowledge Acquisition Approach to GDSS Design

We take a unique approach to GDSS decision modeling. We base our approach on our laboratory's experience with knowledge acquisition and decision support tools. Using

knowledge acquisition methods could significantly improve the effectiveness of GDSS's by addressing their modeling inadequacies.

Many knowledge acquisition tasks are similar to group decision support system tasks, and many knowledge acquisition tools use decision models (Boose, 1991). In both cases, decision information must be elicited, represented, used to gain insight about a decision, verified, and validated. Decision models in both areas guide the type of information that should be captured. Users iteratively refine decision models as they expand and test them. Eliciting and using knowledge from multiple experts is similar to the problem of gathering information in a group setting.

The field of knowledge acquisition has developed powerful methods that help users directly build, test, and refine models. We focus on representations and analysis methods used by successful knowledge acquisition tools that we and others have built.

Knowledge acquisition for decision support follows these steps:

1) Identify a problem solving model by examining features of the problem. Section 1.1 describes knowledge acquisition as a *modeling activity*.
2) Define knowledge types and roles. Important roles in a GDSS include information, preferences, and alternatives. Section 2.2 discusses these elements of a decision.
3) Section 2.3 describes the role of *mediating representations* that effectively communicate important problem aspects.
4) Section 2.4 discusses the cost - benefit tradeoff of building more complex decision models.
5) Design or adapt elicitation and analysis techniques. Section 3.0 describes the components of our decision model.

2.1 Knowledge Acquisition as Modeling

Recent work in knowledge acquisition for knowledge-based systems has emphasized that the creation of knowledge bases is a constructive *modeling* process, and not simply a matter of expertise transfer or knowledge capture. Clancey (1986) stated that knowledge acquisition is the familiar scientific problem of trying to create a model where, in principal, none existed before (e.g., Bradshaw and Boose, 1990; Clancey, 1984, 1990; Cox, 1991; Ford and Adams-Webber, 1991; Gruber, 1990; Musen, 1988; Shaw and Woodward, 1990; Wielinga, Akkermans, Schreiber, and Balder, 1989).

We use knowledge acquisition methods to guide our thinking about the problem of building decision systems. We use related concepts to define problem types and the roles knowledge can play — the elements of a decision. We fill in the model gaps in group decision support systems using successful approaches from our knowledge acquisition experience.

A model-based description of the problem given in a form the user can intuitively understand has many advantages. One important one is that it can serve to mediate communication between users of the system, helping them articulate and understand the broader, higher-level problem context. The section below on mediating representations discusses some of these advantages.

Knowledge-based system researchers have defined many application problem and problem solving method taxonomies (Boose, 1991). These models establish and control the sequence of actions required to do a task. For example, *Analysis* problems involve identifying sets of objects based on their features. *Synthesis* (generative, or constructive) problems require that a solution be built up from component pieces or subproblem solutions (Figure 2). Associating a problem with a category helps define the kinds of knowledge, elicitation, analysis, and reasoning that we need to reach a decision.

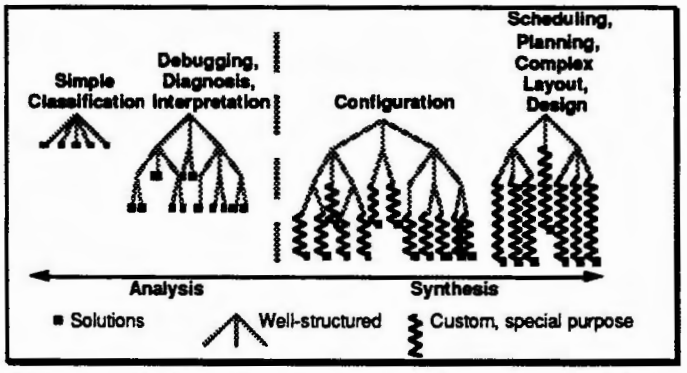


Figure 2. Problem classification techniques borrowed from knowledge-based systems help define decision support models.

Ideally, decision modeling tools should also support the entire decision life-cycle, from initial conceptualization to finished implementation of a solution. Each phase of the life-cycle, however, poses its own problems and has its own requirements. Many of the problems associated with information acquisition and maintenance stem directly from the inadequacies of representations used at various stages in the development of decision models. Our representations and tools should support a smooth evolution of the model from an easily communicated, unconstrained, conceptual statement of the problem to an unambiguous specification of decision reasoning system. This suggests a requirement that decision tools accommodate the changes in representation that may accompany successive stages in model construction: from mental models to increasingly refined conceptual models via elicitation and analysis techniques, and eventually, from these highly elaborated models to an operational decision model information base via formalization and implementation procedures (Shaw and Woodward, 1989).

2.2 Elements of a Decision

A complete decision model, containing relevant items of problem-solving knowledge and their interrelationships, constitutes the *decision basis* (Howard and Matheson, 1984). The decision basis combines *information*, *preferences*, and *alternatives*. Information consists of the knowledge about the problem. Preferences are factors that determine the desirability of an alternative, such as cost, effectiveness, or risk. Alternatives are the possible solutions for the decision. Figure 3 shows these three types and some important subtypes of knowledge.

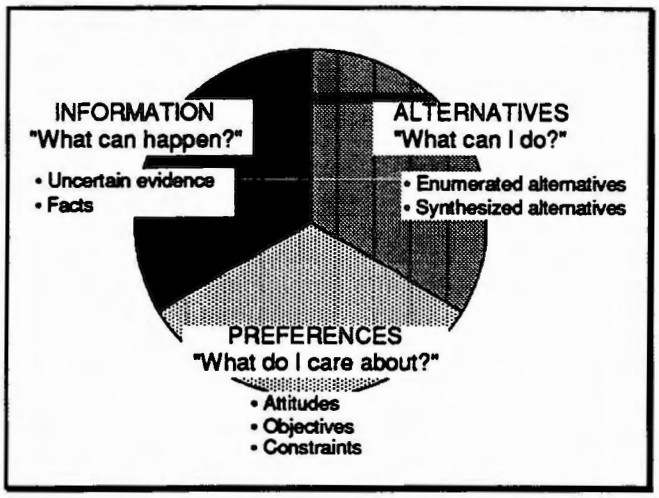


Figure 3. Building blocks of a decision include information, alternatives, preferences, and their interrelationships.

2.2.1 Information

One important part of decision knowledge is *information* about actual or possible circumstances in the world that affect the decision. For example, we might want to know the likelihood of the timely delivery of a critical resource before deciding whether to fund a project. Finding out about a predicted delay may change our decision.

It is useful to think of information as being of two types: *uncertain evidence*, which are of statements believed to be true with some probability, and *facts*, which are statements believed with certainty. Facts are a special case of uncertain evidence in which the mapping onto the space of probability distributions happens to be concentrated on a single point. However, by making this distinction, we can often formulate a problem with additional clarity and computational efficiency.

We can have incomplete information about the present and uncertainty about the future. The decision may involve high degrees of risk. These problems affect the decision process.

2.2.2 Preferences

Preferences describe the multiple, often competing goals that we value as outcomes of a decision. The strength of our preferences motivates us to make a careful choice among alternatives. If we really didn't prefer project success to failure, or prefer spending less money to spending more money, any effort we put into making a decision would be wasted.

It is useful to distinguish between *direct* and *indirect* preferences. Direct preferences relate to things we value for their own sake. Indirect preferences have no intrinsic value except as they relate to direct ones. For example, when purchasing a car, most people place a direct value on cost. However, fuel economy is usually only indirectly valued because of its contribution to overall cost.

There are three different types of preferences: *attitudes*, *objectives*, and *constraints*.

1. **Attitudes** consist of items such as *time preference* (e.g., the desire to receive good outcomes sooner rather than later) and *risk attitude* (e.g., the desire to pursue a possibly less profitable policy to avoid risk).
2. **Objectives** relate to the significant positive and negative consequences of the alternatives that a decision maker wishes to maximize or minimize. In decisions with multiple objectives, we must find methods of quantification and joint measurement (i.e., commensuration) to make tradeoffs between them.
3. **Constraints** specify the conditions for maximizing objectives. They define the limits of the space of acceptable outcomes. Constraints may be matters of definition (e.g., "There is $2 million budgeted for the project this year") or of principle (e.g., a moral belief that precludes consideration of alternatives such as corporate spying to speed project development). We can relax many hard constraints. For example, if we determine that a contemplated project cannot be finished within its original time and budget constraints, we may decide to increase the budget and hire additional personnel to meet the deadline.

Explicit modeling of preferences leads to the identification of important criteria, the ability to handle complex tradeoffs, and the ability to reach a difficult consensus.

1. **We can address unique tradeoffs.** The model represents preferences and tradeoffs explicitly so they can be seen by all and modified directly.
2. **We can evaluate the effects of pieces of evidence.** By performing *sensitivity analysis* we can examine whether a particular piece of information in favor of a project alternative will have any real effect on the allocation decision.
3. **We can measure the value of obtaining additional information.** We can ask, "What is the most I should pay to gather intelligence about a competitor's project in this area?"
4. **We can determine the value of controlling an uncertain variable.** To guarantee the timely delivery of a critical resource we could assess the value of acquiring or merging with another company.
5. **We can measure and use risk attitude and time preference.** In some situations, it is worthwhile to model the company or department's attitude toward risk before making a decision. Time-critical situations may also require explicit modeling of the risks and benefits of delaying or hastening the course of a project.
6. **We can express recommendations in value terms.** It may be important for a system to not only recommend the favored budget plan, but also to estimate its total cost or benefits in some meaningful unit of measurement. We can address the quantification and joint measurement of "intangibles" such as company image and quality of worker life.

2.2.3 Alternatives

Alternatives are the courses of action that may be recommended consistent with the decision basis. After hearing an economic report (information) and determining the effect of the report on the success of the projects (preferences) we will choose between projects (alternatives).

Often alternatives are synthesized from constrained components. For example, including certain components of a design may preclude other components. Possibility tables, discussed below, represent complex inter-component constraints and help users synthesize new solutions.

2.3 Mediating Representations

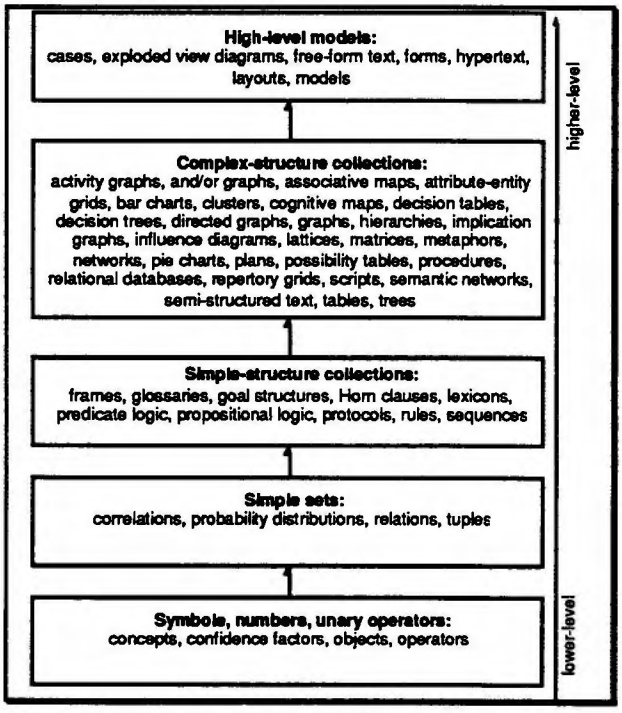


Figure 4. Effective mediating representations make the important elements of a decision explicit.

Effective mediating representations are critical to the success of both knowledge acquisition tools and group decision support systems. They are the users' window on the decision model. Many knowledge acquisition tools achieve success in part through the development and adaptation of good mediating representations (Figure 3).

We use the term mediating representation to "convey the sense of ... coming to understand through the representation" (Johnson, 1989). The design of a mediating representation should be optimized for human understanding. Effective mediating representations ease participant involvement in a group setting.

Winston (1984) says effective representations make important things explicit and hide unnecessary detail. They expose natural constraints, facilitate computation, and are complete and concise. The choice of representation can have an enormous effect on human problem solving performance (Larkin and Simon, 1987). As an example, consider that we can represent numbers as Arabic numerals, Roman numerals, or as bits in computer memory. While each of these forms are logically equivalent, they are not the same in a practical sense. It is much more efficient for a computer to multiply numbers represented as bits than as numeric symbols. Similarly, from a human perspective, it is easier to do multiplication with Arabic numerals than with Roman numerals or binary numbers.

A good mediating representation can simplify modeling processes by providing a medium for users to model their valuable but difficult-to-articulate knowledge about an explicit external form. The mutual development of a representation supplementing the exchange of information between participants promotes and enriches communication, leading gradually to a shared understanding of the emerging conceptual model of the domain (Norman, 1988, 1991). Mediating representations enable participants to cooperatively build decision models. Mediating representations may also simplify maintenance and explanation by enabling users to explore the conceptual domain model without resorting to low-level representations (e.g., C code, Lisp, rules, pure text).

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 do some task (Gruber, 1989; Karbach, Linster, and Voß, 1990; Klinker, 1989). The problem-solving model defines the type of knowledge applicable within each step, thereby making explicit the different roles knowledge plays. Once these roles are defined, we design representations and procedures needed for acquiring each type of knowledge. Research on mediating representations has generally attempted either to improve the computational expressiveness of human-efficient representations (e.g., repertory grids, decision trees, hyper-text) or to improve the learnability of computationally powerful ones (programming-by-example, fourth-generation languages).

Automated knowledge acquisition tools are beginning to incorporate effective mediating representations. These tools tend to adopt one of two approaches. Either they contain interfaces that bear a close resemblance in appearance and procedure to the original manual task—for example, cancer-therapy protocol forms in OPAL (Musen, 1988) and engineering notebooks in vmacs (Sivard, Zweben, Cannon, Lakin, and Leifer, 1989), or they rely on some easily-learned, generic knowledge representation form—for example, object hierarchies and repertory grids in DART (Boose and Bradshaw, 1987; Boose, Shema, and Bradshaw, 1989, 1990).

2.4 Decision Model Costs and Benefits — Applying Just Enough Effort to a Decision

There is a tradeoff between the costs and benefits of building more complex decision models. Ideally, we should apply just enough effort to solve a problem with the needed level of accuracy. Finding this level is usually an iterative process. We wish to start with simple models and expand them in critical areas. Sensitivity analysis can help identify these areas. A simple budget model that starts with lists of projects may expand to include explicit criteria, interacting constraints, and risk management factors.

Decision analysis (Howard and Matheson, 1984), for example, is a rigorous methodology that involves eliciting

numbers that represent uncertainty, value, and risk. Many decisions do not require this level of rigor. Sometimes the cost of such analysis is not justified by the small size of the problem; sometimes the problem seems so simple that the best alternative is obvious and we can act immediately (Cyert and March, 1963; March, 1978). For personal decisions involving ethical issues, such as whether to consider an abortion, the methodology itself may be largely inappropriate (Levi, 1986).

Even among complex decisions, analysis requirements vary: different problems demand different amounts of emphasis on modeling different kinds of problem-solving knowledge. Complex decision problems require both information and preference modeling for the system to enable insight and effectively formulate recommendations. Grids, for example, enable new forms of analysis in a group setting. These analysis methods can be keys to gaining understanding and insight about a problem.

Table 1 displays a set of problems as a function of the amount of uncertainty and the complexity of preference issues.

We should determine the complexity of the model, then, by examining the types of information needed to solve the problem, the precision and accuracy needed, the time and resources available, and the relative importance of the decision.

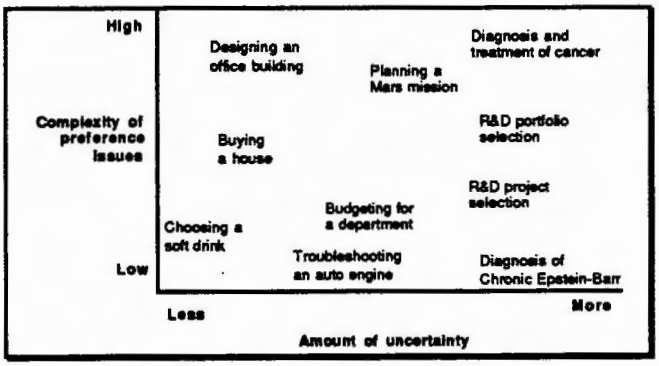


Table 1. Problems with higher uncertainty and complexity require more complex decision models.

3.0 Building a Stronger Model for a GDSS

In this section we develop a GDSS decision model that incorporates aspects of many tools and techniques. First we discuss DDUCKS, our integration testbed. Then we build the decision model by adding knowledge roles, methods, and mediating representations to solve different kinds of problems.

3.1 An Integrating Infrastructure

As part of a project entitled Design of Information Systems (Benda, 1990) we are defining an "open architecture" integrating environment. We call this environment DDUCKS (Decision and Design Utilities for Comprehensive Knowledge Support). DDUCKS is the underlying infrastructure for our group decision support workbench. Previously we used individual components of DDUCKS for several applications. Now we are exploring how the components can work cooperatively to help users solve complex decision problems (Bradshaw, Covington, Russo, and Boose, 1990, 1991).

An underlying intermediate representation in DDUCKS stores decision information. DDUCKS transforms this information between internal components and ultimately between mediating representations. Users seeing information in different forms triggers insights since each representation makes different information explicit while hiding other information.

Sometimes DDUCKS needs additional information to assist the transformation. For instance, gIBIS criteria must receive range, type, and weight information before their use in a repertory grid. The most difficult transformation — between grids and influence diagrams — is discussed in detail in Bradshaw and Boose (1990) and in Bradshaw, Covington, Russo, and Boose (1990).

We attempt to use a common vocabulary for knowledge roles and processes across all components of the model. For instance, we use the term *criterion* for: the decision literature term *decision variable*, the statistical term *dimension*, the psychological terms *construct*, *characteristic*, and *trait*, the decision table term *attribute*, the IBIS term *argument*, and the design term *rationale*. Designing the common information structure underlying these terms was a large part of the effort of building our integrated model. Understanding these relationships simplifies the problem of representing and transforming information for different modeling tasks.

While this paper focuses on decision model content, DDUCKS contains process management tools. These could be used to model and execute the process of using the group decision support workbench. They could also model the decision's business enterprise context (Bradshaw, Holm, Kipersztok, Nguyen, and Covington, 1991).

3.2 Building the Model

The first section below discusses a simple brainstorm-and-score model where we generate and rank alternatives.

In later sections we add additional knowledge roles and methods to the brainstorm-and-score model. We build knowledge roles from the elements of a decision — information, preferences, and alternatives. Each new section describes the portions of the model needed for solving certain additional kinds of problems.

Described along with each new decision model component are its knowledge roles, application problem characteristics and examples, processes and methods used, example mediating representations, and group use issues.

3.2.1 Part 1: Brainstorm and Score — Rapid Decisions Using Implicit Criteria

A simple decision making process includes generating and ranking alternatives (Figure 5). This process is valuable for making rapid decisions or decisions where it is not beneficial to build any more of the underlying model (Figure 6).

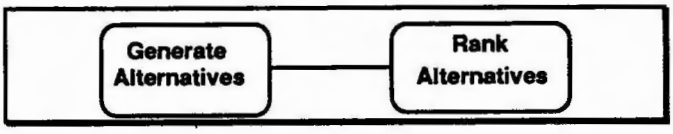


Figure 5. The processes of generating and scoring alternatives are at the heart of most decision problems.

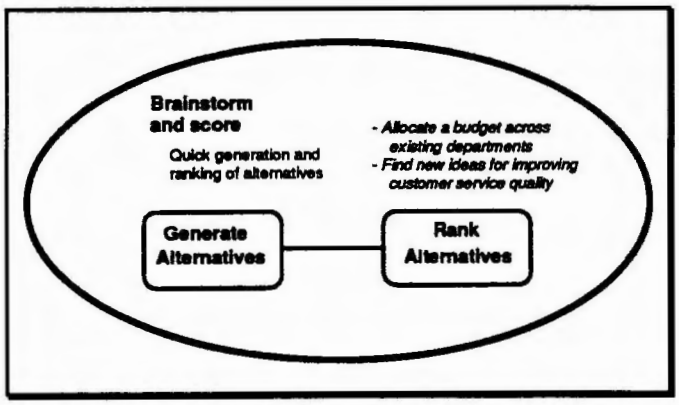


Figure 6. A fast and simple way to reach a group decision is to brainstorm alternatives and score them. Tools such as GroupSystems, TeamFocus, and VisionQuest support individual and group display of decision results.

Description

A team that wants to increase the quality of customer service might use a brainstorm-and-score system as follows:

1. Participants use **brainstorming** to generate ideas for the question, "What inhibits our customers from being fully satisfied with our products and our services?" Participants enter and view ideas anonymously, in parallel.
2. **Idea organization** tools help the team pull out key inhibitors to achieving customer satisfaction from the brainstormed list. Participants organize comments from brainstorming into a list of key issues.
3. Each group member **votes**, rating each inhibitor on its potential for improving customer satisfaction.
4. **Brainstorming** again generates ideas on how to overcome the worst inhibitors.
5. Finally, **electronic commenting** helps form team policy — "To improve customer satisfaction, who in this room should do what and when? What are the next steps?" Again participants enter and view ideas anonymously.

Section 1.1 summarized the applications, benefits, and group uses of systems supporting this model.

Processes, Methods, Knowledge Roles

This section includes typical processes, methods, and knowledge roles for brainstorm-and-score systems.

Generate requirements — Define the needs that help identify the problem.

Define problem — Define the problem and a process for solving it. Some tools have *templates* or *agendas* that define process paths through sets of tools.

Organize problems — Use *outlining* to group problems in categories.

Define participants — Define the members and their roles (such as participant or facilitator) in the problem-solving process with *rosters* or *group lists*.

Organize participants — Categorize participants by session or problem type.

Generate alternatives — Identify potential solutions for a problem through *brainstorming* or *structured commenting*.

Organize alternatives — Organize alternatives in categories with an *outlining* or *grouping* tool.

Weight (score) alternatives — Score solutions for a problem individually or by group. Common methods include placing items in order (*ranking*), assigning *rating scores*, *voting* (yes / no / abstain), *selecting* several alternatives from a list, and *allocating fixed resources* among alternatives.

Generate and view team results — Display *consensus* results and *variation* within the group.

3.2.2 Part 2: Structured Outlining — Linking Alternatives and Criteria

We can increase the utility of the group decision model by using explicit criteria (Figure 7). The relationships between alternatives and criteria and the criteria themselves can have simple or complex structures.

The gIBIS model uses unstructured criteria (text entries) in structured relationships (tagged outlines) (Conklin and Begeman, 1988, 1989) (Figure 8). Although designers use gIBIS primarily for design rationale capture the structure applies equally well for developing other kinds of decision discussions in a structured format.

A gIBIS-like model addresses the problems of **documenting running discussions** and **making criteria explicit** (mentioned in Section 1.3).

Description

We have adapted gIBIS's vocabulary to blend with our decision model. gIBIS's *issue* is the overall problem; *positions* are alternatives; *arguments* are criteria.

The term gIBIS means "graphical issues-based information system." gIBIS initially captured design rationale. The system provides a language and browser that allow individuals to add criteria for and against alternatives for certain problems (Figure 9).

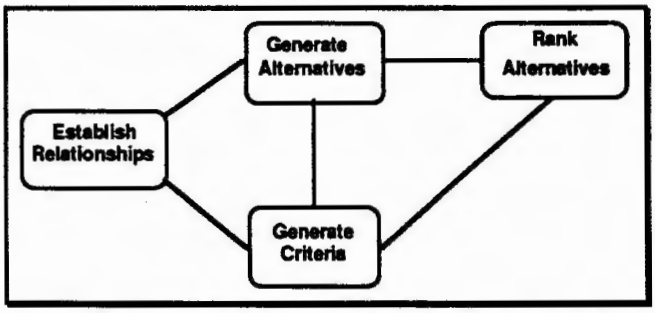


Figure 7. Sometimes making criteria explicit helps the group reach a better decision. Explicit criteria allow alternatives to be examined in detail. They can also illuminate areas of agreement and disagreement.

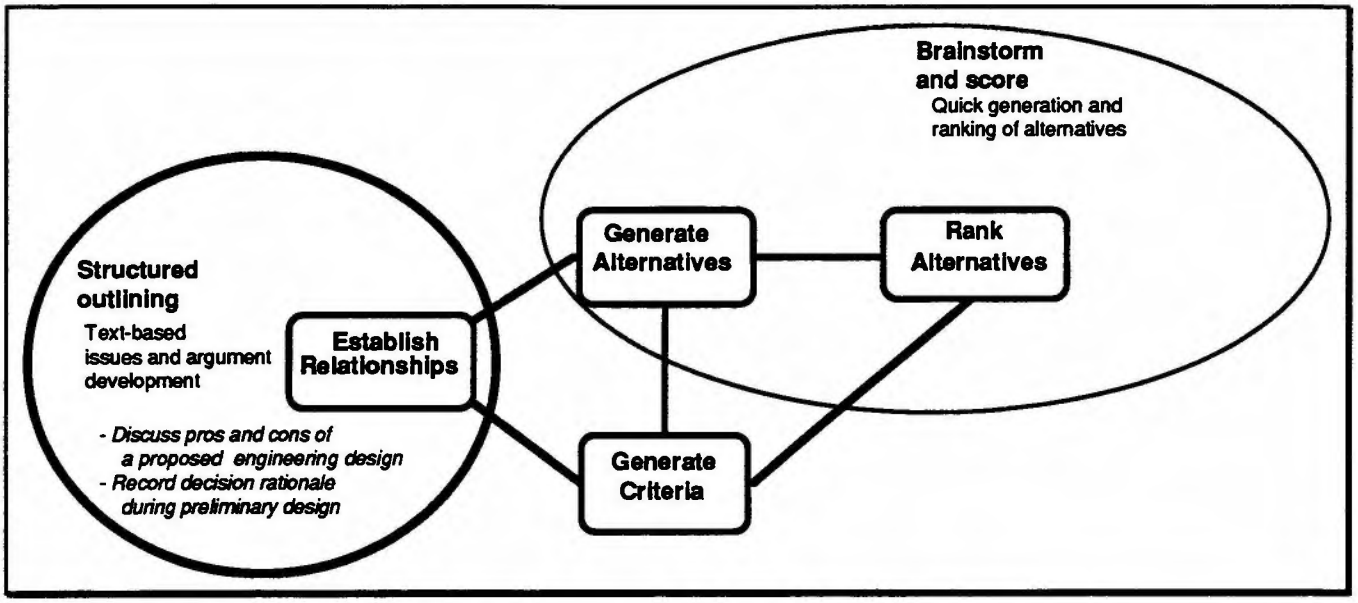


Figure 8. Group members can record detailed arguments for and against alternatives with structured outlining tools such as gIBIS.

The mediating representation is an outline where *indentation* shows relationships between alternatives and criteria, and *tags* show other aspects of criteria, such as whether criteria are for or against a position (Figure 10). This language is straightforward to learn and read. gIBIS provides a browser that shows the relationships graphically.

Since gIBIS records arguments as they evolve the model also captures aspects of change over time.

Other tools such as SYBIL extend the gIBIS model (Lee, 1990). SYBIL helps manage dependencies, uncertainty, viewpoints, and precedents by extending the relationship and tagging structure.

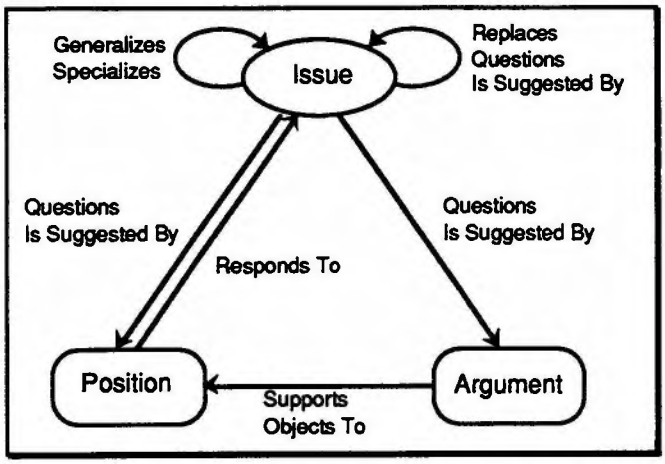


Figure 9. gIBIS represents several types of criteria for alternatives.

```
*I:  Which processor should be used?
 ?P:  Processor A.
  AS:  Fast
 *P:  Processor B.
  AS:  Already in use, thus cheaper.
 -P:  Processor C.
  AO:  Won't be available in time.
```

Figure 10. Indentations in IBIS text files represent hierarchical relationships. Issues are labeled with "I", positions with "P", supporting arguments as "AS", and objecting arguments as "AO". "?" means that the position is still open; "*" means that the position is resolved; "-" means that the position is rejected (Yakemovic and Conklin, 1990).

Applications

The IBIS method is useful for design and planning (Kunz and Rittel, 1980). It helps in structuring exploratory thinking in groups, addressing group decision support, conversational structuring, and management of group memory. Designers in group projects used gIBIS during document analysis, requirements and design meetings, and personal brainstorming (Yakemovic and Conklin, 1990).

Processes, Methods, Knowledge Roles

This section includes typical processes, methods, and new knowledge roles for gIBIS.

Define the problem — Identify the problem issue and distribute it to the group.

Develop alternatives — The group develops position alternatives, adding them to the information base.

Develop criteria — The group advances arguments for and against alternatives in a running format.

Generate document — The group generates a final document showing accepted, rejected, and open alternatives.

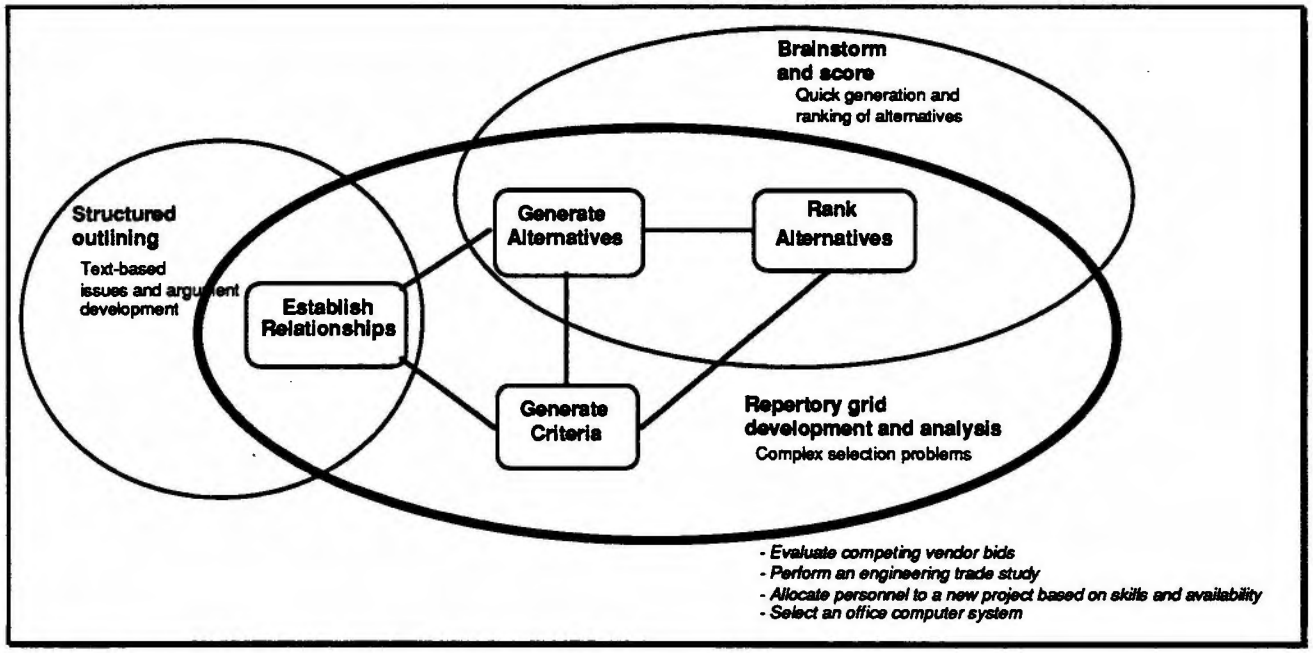


Figure 11. Adding more structure to the relationships between alternatives and criteria enables more sophisticated analysis of the decision information. Repertory grids represent information in a rich tabular form that aids statistical analysis and use of machine learning methods. Tools such as DART augment grids with constraints and preferences .

3.2.3 Part 3: Grids — Linking Multiple Alternatives and Criteria

We can further increase the utility of the group decision model by adding structure to the relationships between criteria and alternatives and by adding structure to the criteria themselves. Explicit criteria in a *grid* format show relationships between alternatives and enable analysis of decision information. Static and dynamic analyses measure a grid's problem-solving power. Other analysis tools measure the independence and completeness of the alternative and criterion space. Defining criteria and later analyses can be critical parts of the decision modeling process (Figure 11).

Building and analyzing grids address the problems of defining **complex criteria** and measuring their effect (mentioned in Section 1.3). Decision methods associated with grids address some of the problems of representing criteria **minimums**, **maximums**, and **ranges**.

Description

DART (Design Alternatives Rationale Tradeoffs) is a repertory-grid-based knowledge acquisition tool. We originally developed DART for NASA as part of an effort to capture design knowledge for the Space Station Freedom program (Boose, Shema, and Bradshaw, 1990). Similar tools and concepts have been under development at The Boeing Company for many years (Boose, 1984, 1985; Boose and Bradshaw, 1987; Boose, Bradshaw, Kitto, and Shema, 1989; Boose, Shema, and Bradshaw, 1989). DART contains elicitation, analysis, representation, and inference methods derived from personal construct theory (Kelly, 1955). Other researchers have contributed elicitation and analysis techniques to grid methods for knowledge acquisition, notably Shaw (1979), Gaines and Shaw (1981, 1990), Diederich, Ruhman, and May (1987), Garg-Janardan and Salvendy (1987), Gaines (1987), Shaw and Gaines (1987), and Ford, Stahl, Adams-Webber, Novak, and Jones (1990).

Knowledge acquisition tasks performed by DART include eliciting criteria, decomposing problems, combining uncertain information, incremental testing, integration of data types, automatic expansion and refinement of decision information, use of constraints during decision reasoning, and providing process guidance. DART interviews users and helps them analyze, test, and refine the model. The system represents, analyzes, and reasons with information from multiple users separately or in combination. DART derives decision results by propagating information through alternative and criterion hierarchies.

DART uses tools to elicit and structure information about alternatives, criteria, constraints, and preferences. Static and dynamic analysis tools in DART help determine the adequacy of the decision model and help focus users' attention on parts of the model needing further refinement.

DART also uses *machine learning* techniques to help refine the decision model. Group members may use induced information to further refine the decision model. Machine learning takes place in DART in interactive and automatic forms. Interactive forms include implication generation, analysis, and review. Automatic forms include strategies embedded in the reasoning mechanism and methods to automatically improve the decision model.

Applications

Figure 12 shows a repertory grid for a NASA Space Station Freedom problem. Two teams verified a module's location on the current design configuration. We used DART to elicit and analyze information from both teams separately and then combined the grids together to reach a solution.

Repertory grids have been applied to hundreds of kinds of problems. In Boeing we applied them to aircraft design and

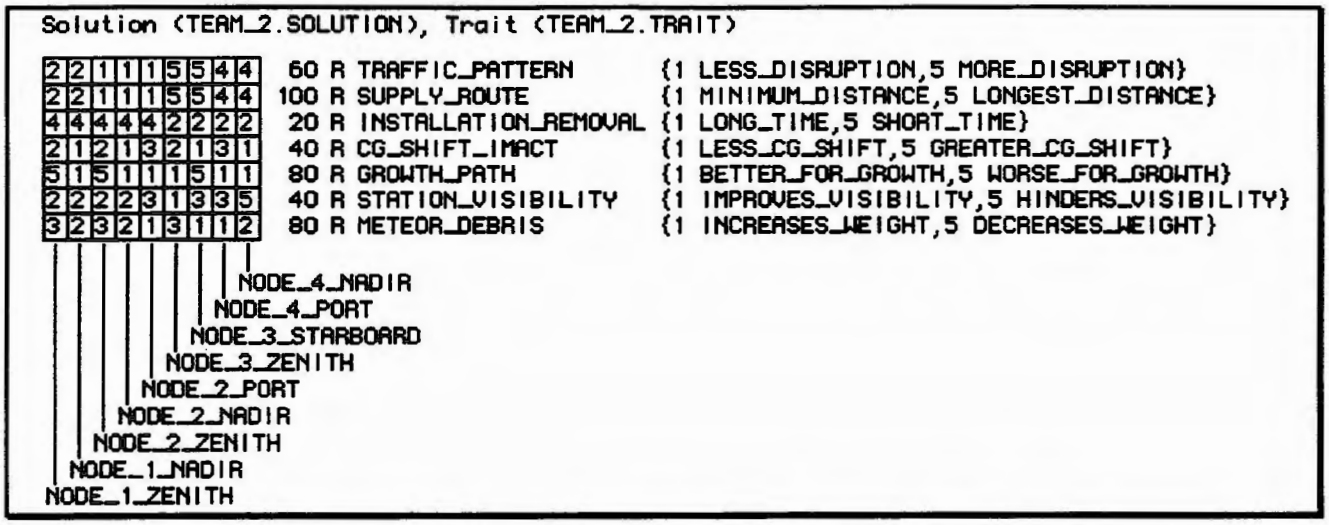


Figure 12. One design team's simple repertory grid for an engineering trade study problem. DART combines information from several designers to show consensus and dissension.

manufacture (e.g., aerodynamic analysis, noise certification, documentation updating, materials technology, plant location), computer systems maintenance and design (e.g., hardware and software selection, product design and impact, statistics interpretation), defense and space system design (e.g., Space Station Freedom trade studies, composite materials, energy control systems, image analysis), procurement and sales (e.g., vendor selection, customer product matching, services advising, sales support), and organization support and administration (e.g., employee evaluation, hot-line help, office automation, personnel selection, training) (Boose, 1988). Table 4 shows examples of repertory grid use at Boeing.

Aircraft Design and Manufacture
Aerodynamic Anal. - Geom. Adv.
Aerodynamic Anal. - Front End Advisor
Aerodynamic Anal. - Back End
Aircraft Fault Isolator
Airplane Mass Property Est. Risk Analyzer
Airplane Design Flutter Analyzer
Airplane Noise Certification Advisor
Automated Numerical Control - Cutter Consultant
Automatic Flight Controls Diag. Aid
B1 Diagnostic Consultant
Bond Durability Consultant
Documentation Update Consultant
Failure Modes and Effects Analyzer
Finish Advisor for Design Engineers
Finish and Corrosion Control Consultant
Flight Controls Human Factors Assistant
GT-STRUDL Structural Analysis Adv.
Jet Engine Diagnostic Aid
Jet Engine Fault Isolator
Jet Engine Manual Advisor
Materials Technology Advisor
Molded Rubber Seal Advisor
Parts Quality Control Consultant
Plant Location Selector
Propulsion System Advisor
Resin Advisor for Composite Parts
Rivet Selector
Structural Analysis Software Selector
Transport Airplane Config. Selection Adv.
Velocity Analysis Advisor

Computer Systems Design and Maintenance
Business Computing Needs Advisor
Business Graphics Package Con.
Comp. Comm. System Bug Adv.
Computer Languages Consultant
Database Management System Consultant
Graphics Package Advisor
Micro Computer - Needs Analysis
Micro Computer - System Consultant
Micro Computer - Workstation Config. Adv.
Micro Computer Trouble Shooter
Product Design and Impact Advisor
Product - Comparative Analysis
Product Marketing Advisor
Prog. Language Applications Adv.
Programming Language Eval. and Selector
Questionnaire Development
Rel. Database Construction Adv.
Software Management Consultant
Software Quality Advisor
Software Release Advisor
Software Services Advisor
Statistics Interpretation
Statistics Package Use Advisor

Defense and Space System Design
Carbon Dioxide Removal (Space Station Freedom)
Circuit Break Interface (Space Station Freedom)
Composite Materials Advisors
Drawing Formats (Space Station Freedom Program)
Energy Control System Model Eval.
Experiment Configuration
Helicopter Avionics Diagnostic Aid
Helicopter Hover Advisor
Helicopter Stick Position Advisor
Helicopter Vibration Diagnostic Aid
Image Feature Analysis
Navigation System Advisor
Potable Water System (Space Station Freedom)
Pressurized Logistics Module Placement (Space Station Freedom)
Shuttle Experiment Config.
Space Station Window Materials Con.
Visual Target Identifier

Procurement and Sales
AI Vendor Consultant
Contract Award Advisor
Customer - Product Matching
On-line Services Advisor
Sales Support Information Analysis
Technical Sales and Services Consultant

Organization Support and Administration
Delphi Group Information Gathering
Employee Evaluation
Hot-line Consulting Aid
Management Motivation Analyzer
Negotiations - Unreasonable Offer Response
Negotiations Advisor - Info. Seeking
Negotiations - Joint Gain
Negotiations Advisor - Obtaining Leverage
Negotiation Arbitration Environment
Office Automation System Advisor
Office Automation Workstation Advisor
Organization Devel. Intervention
Organizational Climate Diagnostician
Personnel Resource Mgt Decision Aid
Personnel Technology Matching
Personnel Selection Consultant
Research and Devel. Lab Site Advisor
Research Project Priority Advisor
Survey Reporting and Analysis
Task Priority Manager
Tax Regulation Consultant
Training Course Evaluator
Training Curriculum Advisor

Table 4. Repertory grids have a wide application range.

Processes, Methods, Knowledge Roles

This section includes typical processes, methods, and new knowledge roles for repertory grids.

Generate alternatives — Potential solutions for a problem form the horizontal grid axis. *Similarity analysis*, *implication analysis*, and *table completion* help generate new alternatives or classes of alternatives not yet represented in the grid.

Weight alternatives — Score solutions for a problem. This may be done with *rank ordering* which scores alternatives based on criteria value preferences. Scoring is also possible using *pairwise comparison*. *Similarity analysis* of the alternatives yields a static measure of the criteria's ability to discriminate among the alternatives.

Organize alternatives — Organize sets of alternatives for better analysis, comprehension, and to reduce complexity. *Cluster analysis* suggests possible organizations (Figure 13).

Generate criteria — Make factors affecting alternatives explicit. Criteria definitions include type (nominal, ordinal, interval, ratio), value range, weight, and other information that helps the reasoning process. Many methods help generate criteria. *Triadic* and *dyadic comparison* methods elicit criteria that distinguish between certain sets of alternatives. *Alternative similarity analysis* points out highly similar alternatives that would always receive similar scores during decision making. The user enters new criteria to help further distinguish between them. *Criteria similarity analysis* measures criteria subsumption and independence. *Boundary analysis* encourages users to generate hidden or external criteria that may control a situation. *Laddering* helps generate criteria at higher or lower levels of abstraction to ground problems and to break important criteria into subcriteria. Criteria are also generated during some types of decision model *verification* and *debugging* based on rank order results.

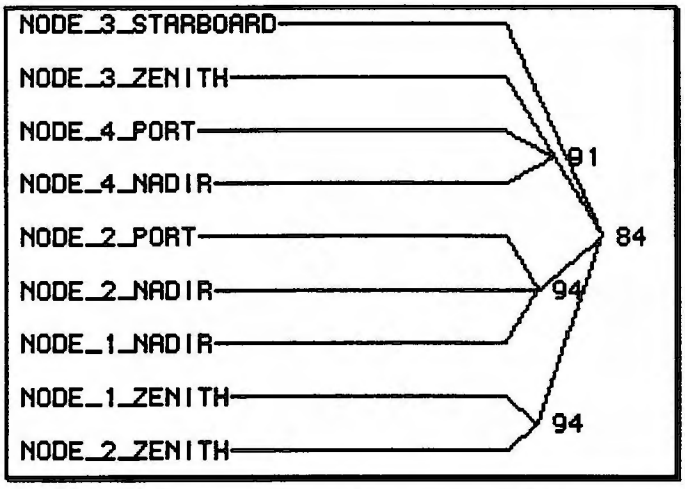


Figure 13. Cluster analysis shows users similarities and helps them decompose or reorganize problems. Users can label numeric junctions and DART automatically subdivides the grid into subgrids.

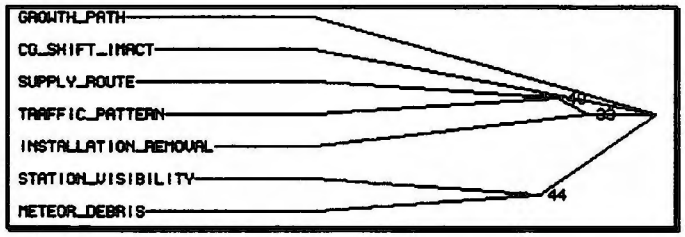


Figure 14. Here cluster analysis is done for criteria for the configuration design problem.

Organize criteria — Organize sets of criteria for better analysis, comprehension, and to reduce complexity. *Cluster analysis* suggests possible organizations (Figure 14). *Laddering* can help organize criteria into networks. *Implication analysis* shows subsumption relationships and other patterns among criteria.

Weight criteria — *Weights* record the relative importance of criteria. *Pairwise comparison* can measure the consistency of weight assignments. A decision tree generation algorithm does a *dynamic cost benefit analysis* of criteria during reasoning. *Sensitivity analysis* shows where to focus on model expansion.

Develop criteria — Assign a criterion type (nominal, ordinal, interval, ratio, cyclic) according to the level of precision needed. *Dialogs* and *value analysis* may suggest criterion types.

Generate repertory grid — Score sets of alternatives against sets of criteria in a table format.

Generate preferences — Express preferences for criteria values (soft and hard constraints). Criteria value preferences help to score and rank alternatives.

Generate generalizations — Make generalizations from information in the grid. *Implication analysis* helps check the accuracy of the grid by using a machine learning technique to generate information at a higher level of abstraction (Figure 15). This also shows criteria independence. Implication patterns can show criteria inconsistencies, ambiguities, and equivalences.

An algorithm developed by Gaines finds implications between criterion values (Gaines and Shaw, 1987; Gaines, 1989). The system uses a repertory grid as a set of examples. Criterion values are viewed as logical predicates, alternatives are the operands of the predicates, and ratings are fuzzy truth values. Graphs and lists show implications and their strengths.

Implications show relationships at higher levels of abstraction that are implied by a repertory grid. If the user disagrees with an implication, DART helps refine the grid. Frequently, the user can think of an exception to the implication (a new alternative) that disproves it. The user enters this alternative, rates it, and the implication strength is reduced appropriately. Sometimes implications point out inconsistencies in the way that the user applies a criterion. In such cases a specialization and generalization dialog (*laddering*) helps decompose inconsistent criteria into consistent sub-criteria.

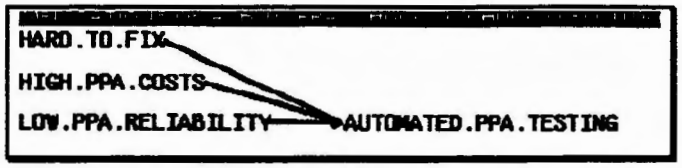


Figure 15. The graph shows the results of an implication analysis. Each of the criterion values HARD. TO. FIX, HIGH. PPA. COSTS, and LOW. PPA. RELIABILITY imply that AUTOMATED. PPA. TESTING is necessary (from an on-board circuit breaker engineering trade study). Participants use inductive generalizations to help refine the model.

Generate constraints — Elicit constraints between alternatives and between criteria. Some constraints limit the choice of alternatives given certain criteria values. Other constraints restrict the values of interdependent criteria during reasoning.

Generate team results — Show combined results of alternative ranking from members of a team (Figure 16). Show team *consensus* and corresponding *dissenting opinions*. DART finds a dissenting opinion by computing a correlation score between each individual and the consensus; the individual with the lowest correlation score is listed as the dissenting opinion. Dissenting opinions show users the *range* of opinion about a decision, not just the top scoring list. Dissenting opinions give decision makers confidence that the top rated alternatives were sound choices or point out areas of disagreement for further exploration.

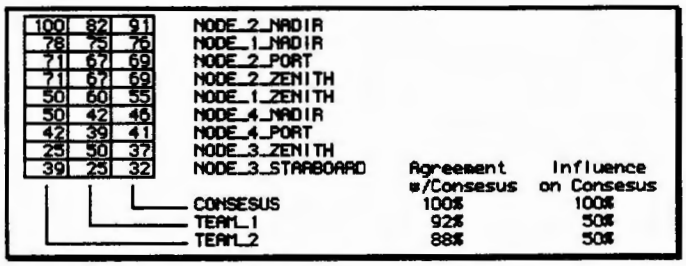


Figure 16. A rank ordering of alternatives shows the consensus and the contributions of individuals.

DART also analyzes and summarizes the *differences* between grids (Figure 17). Other methods point out possible differences in vocabulary and meaning and compute *subsumption* relationships between team members (discussed below).

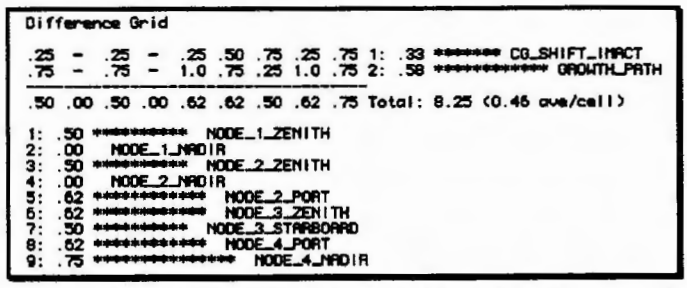


Figure 17. Differences between grids point out areas of agreement and disagreement. Here both teams agree most about the alternatives NODE_ 1_ NADIR and NODE_ 2_ NADIR but disagree most about NODE_ 4_ NADIR and the criterion GROWTH_ PATH.

Evaluate results — Use static and dynamic analyses to evaluate score results and improve the decision model:

Static evaluation — Evaluate the *potential* of a decision model. *Similarity analysis of alternatives* measures the ability of criteria to distinguish between alternatives. *Similarity analysis of criteria* measures the coverage of alternatives and measures criteria subsumption and independence. *Implication analysis* looks for accuracy of information at higher levels of abstraction and checks criteria for consistency, ambiguity, and equivalence.

Dynamic evaluation — Evaluate the *results* of a decision model by testing it. Compare results with *expectations* to measure and verify the performance of the decision model. Use this information to generate new criteria to fix specific problems and improve the performance of the grid.

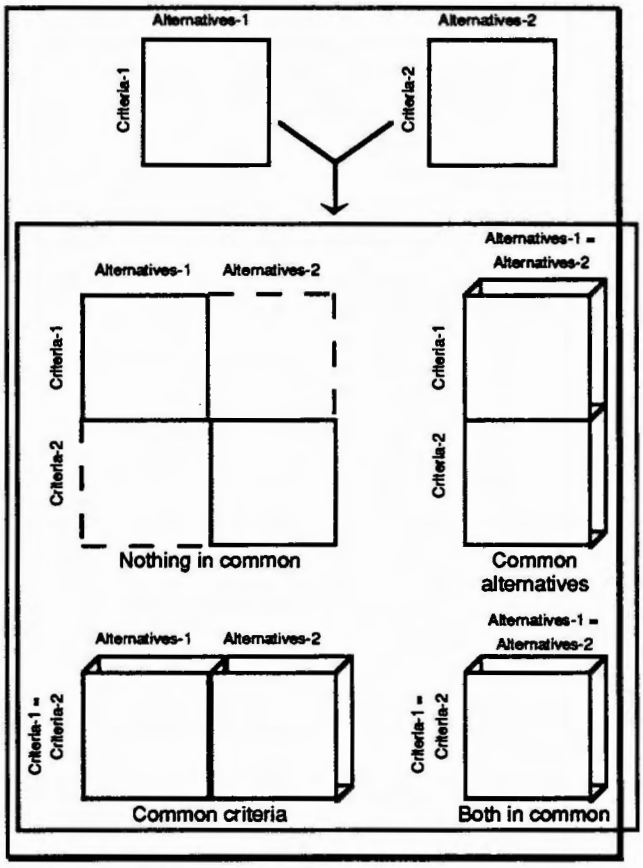


Figure 18. Grids combine in different ways depending on commonality between alternatives and criteria. Group processes help members build common grids that enable further analysis and shared understanding.

Group Use

Grids have been used before in several specialized group settings. Grids combine in different ways depending on commonality between alternatives and criteria (Figure 18). Groups can use iterative methods to proceed from grids where nothing is in common to those with shared alternatives and criteria. Grids with common features can be analyzed in a variety of ways. Shaw used grid comparison techniques to measure differences between pairs of grids and subsumption relationships across groups of grids (Shaw, 1979). Boose adapted this work and added consensus and dissenting opinions when using grids to reach group decisions (Boose,

1986, 1988, 1989; Boose and Bradshaw, 1987; Boose, Shema, and Bradshaw, 1989). Chang used similarity analysis between grids in a dynamic group setting to show similarities and differences as grids evolved (1986, 1991).

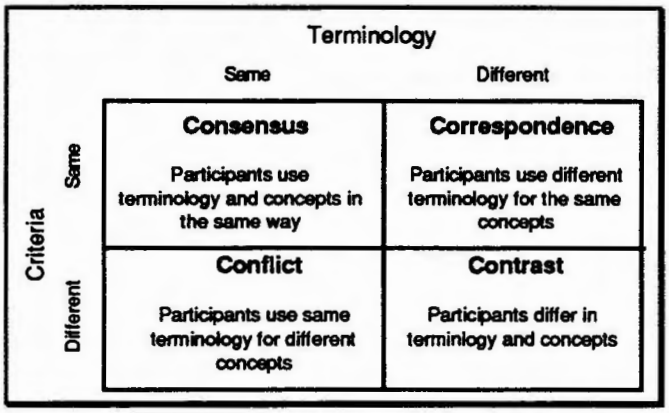


Figure 19. Shaw and Gaines (1989) measured sets of grids and mapped the results based on differences and similarities in terminology and concepts. Their system discovered, for example, that one expert used the criterion "low level data — high level data" in the same way that another used the criterion "nominal data — interval or ratio data". This type of analysis leads to sharing an understanding of vocabulary and concepts between group participants (figure adapted from Shaw and Gaines, 1989).

Shaw, Gaines, and Woodward explored the conceptual space of consensus, conflict, correspondence, and contrast available when comparing grids (Shaw and Gaines, 1988, 1989; Shaw and Woodward, 1988; Figure 19). The facilitator may use several methods for conflict resolution. Boose (1986) describes a structured negotiation technique where participants independently develop grids (divergence), iteratively identify and merge important criteria and alternatives (convergence), and finally reach a decision showing consensus and dissenting opinions (also used in Schuler, Russo, Boose, and Bradshaw, 1990). Shaw and Gaines (1989) describe a detailed method of problem formalization, conceptualization and feedback, exchange and comparison of information, and validation.

3.2.4 Part 4: Possibility Tables — Developing Models of Alternatives

Often a simple text entry is not enough to represent an alternative. Alternatives may need to be synthesized from components that involve interacting constraints. For example, 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 evaluate alternatives for suitability under certain assumptions and by applying criteria. 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 serious errors and costly rework.

Defining and choosing complex alternatives can be an important part of the decision modeling process (Figure 20). Possibility tables handle some of the problems of representing **minimums, maximums, and ranges.** They also address the problems of **restructuring complex alternatives** and **exclusivity** mentioned in Section 1.3.

Description

We developed a stand-alone tool named Canard that helps synthesize alternatives from potentially large search spaces (Bradshaw, Boose, Covington, and Russo, 1989; Shema, Bradshaw, Covington, and Boose, 1990). Canard helps generate and structure complex alternatives in a possibility

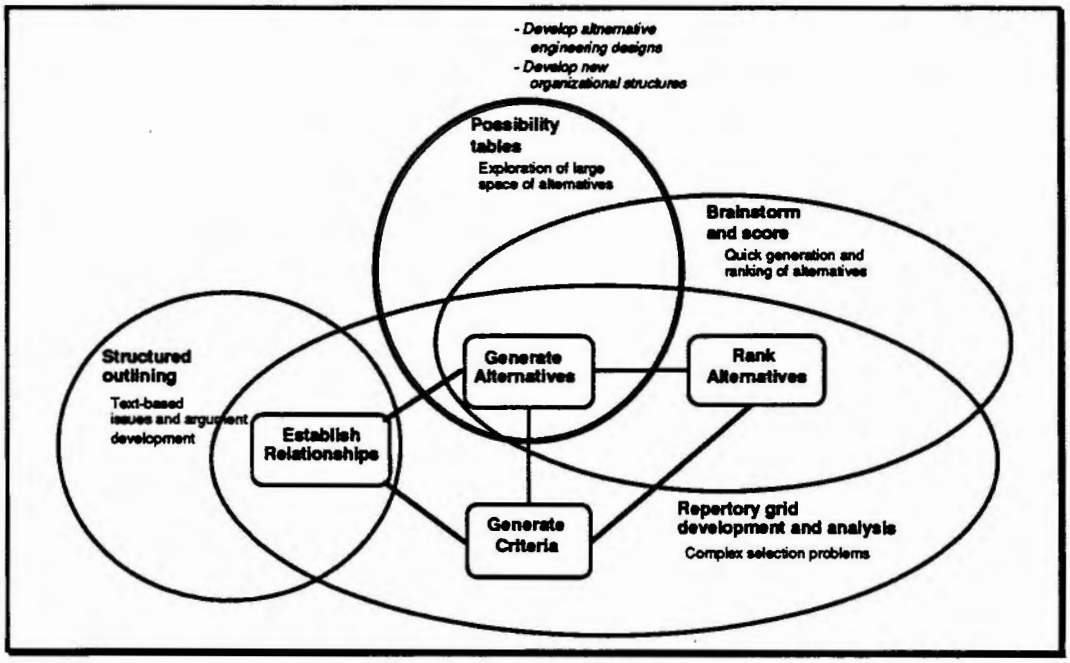


Figure 20. Tools other than brainstorming may be necessary when a problem requires more thorough exploration of a large space of potential alternatives. Possibility tables (also known as morphological charts) help decision makers define paths and constraints through complex components of alternatives.

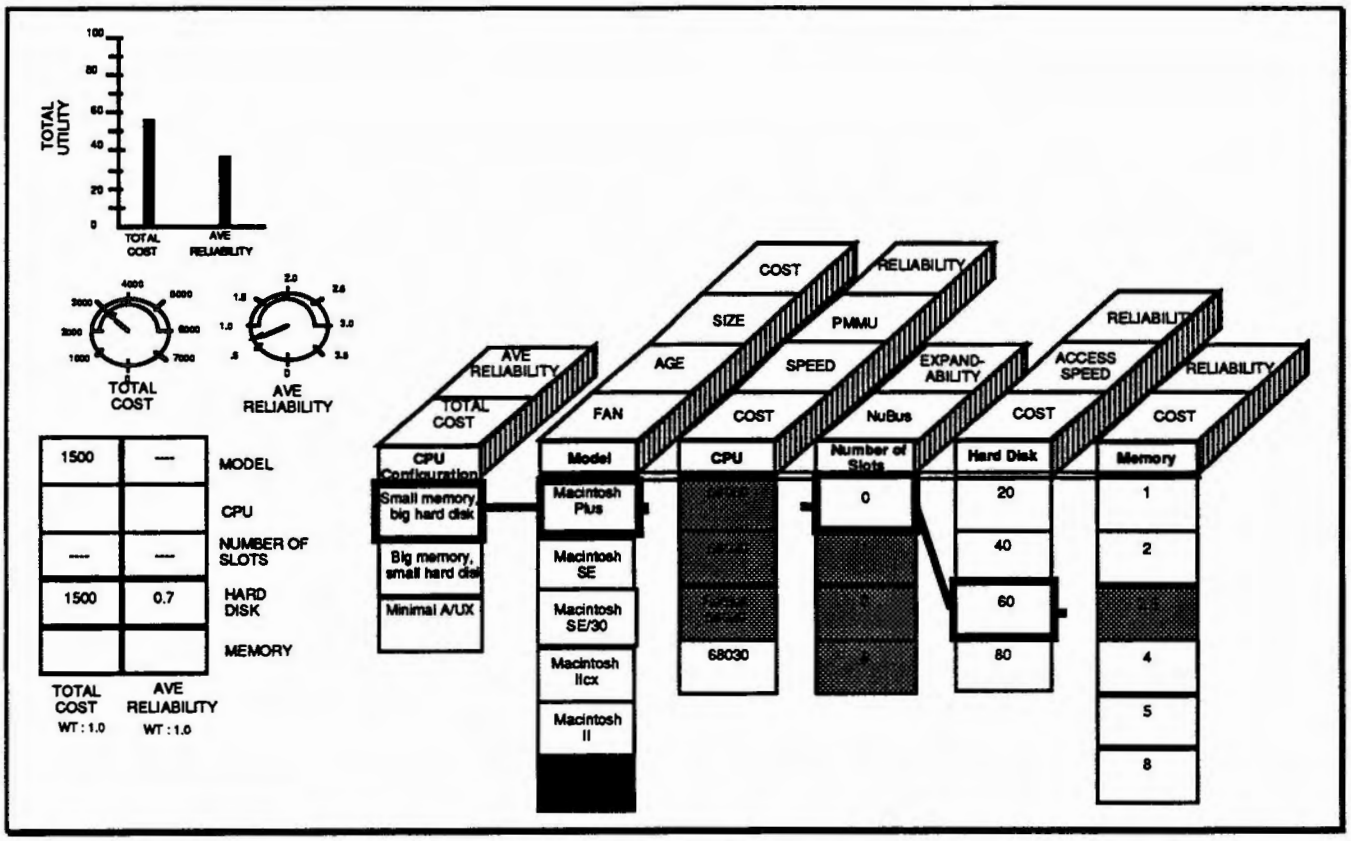


Figure 21. A Canard possibility table for a portion of a network design problem. The thick horizontal line is a partial solution path linking component alternatives. Shaded cells show hard and soft constraints associated with the path. The left-most column shows classes of solutions associated with paths. Repertory grids "plug in" to each column, recording criteria relationships and enabling analysis. Repertory grid reasoning methods help rank the components in a column. Automated methods combine ranking information with constraints to produce a set of best possible solutions.

table. We adapted the possibility table representation from manually developed strategy tables (McNamee and Celona, 1987) and morphological charts (Zwicky, 1969). Decision analysts and designers have used these tools 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).

Canard also enhances possibility tables with constraint-handling tools that reduce the possible solution space and capture important information about the design. The system allows entry of both hard and soft constraints. A designer adds hard constraints to capture information about incompatibilities and interdependencies between component possibilities. During alternatives generation, these hard constraints prevent selection of any incompatible components.

Designers may add soft constraints and utility scores to criteria that characterize generated paths. Preferences for these criteria are the design goals. Conflicts between two or more design goals often require 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 a criterion and map attribute values to the utility of these values. Canard then guides the designer toward possibilities that optimize the tradeoffs between the soft constraints.

Canard provides documentation of the alternative synthesis history. It captures and stores the information used in defining the alternatives. With such a capability, a more complete record of the decision is available for later review and revision.

We are also concerned with helping decision makers 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). DDUCKS maintains links between possibility tables and related repertory grids. By coupling the alternative generation facility to its analysis tools and those in DART, Canard helps users get a better feel for the effects of the constraints on different alternatives and helps them better evaluate the consequences of assumptions and tradeoffs.

Possibility tables could also help with the idea convergence process following brainstorming. Possibility tables could capture the constraints and reasoning behind synthesized categories as items are added. Then as the constraints and reasoning change Canard could automatically manage the associated alternatives.

Applications

Typical applications of possibility tables in Canard include developing alternative engineering designs and structuring new organizations. Figure 21 shows a possibility table for a Boeing network design problem. In this problem Canard stores standard network configuration components and constraints. Customers specify requirements and givens (for example, existing hardware, the number of workstations, or needed applications). The system shows allowable configurations as paths through the component columns. The designer may modify a recommended design (path) based on exceptions or special circumstances. Gauges show running costs and other constraints.

Canard also helps create new problem solutions. The designer builds several component columns, graphically specifying incompatibilities and other constraints between component columns. Combinations of constraints suggest certain partial paths through the columns, which in turn suggest additional components, new columns of components, new constraints, new partial paths and so on. Shema et al. (1990) shows an example of an engineer discovering new robot arm applications and designs. A similar method could help administrators design a new organization structure.

Possibility tables in Canard have a graphical interface which eases creation and change of design information. The designer creates new solution classes and alternatives and defines solution 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.

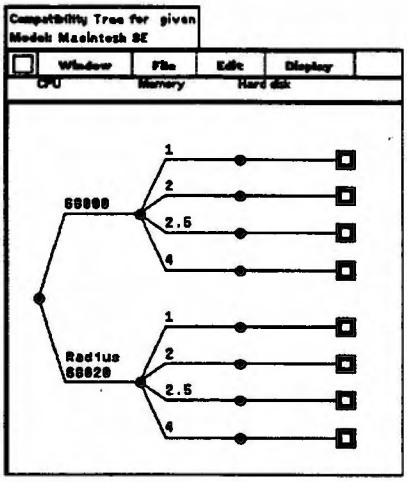


Figure 22. A mediating representation for a partial compatibility tree, showing CPU, memory, and hard disk configurations. Alternative values and criteria value probabilities are recorded in a similar fashion.

Processes, Methods, Knowledge Roles

This section includes typical processes, methods, and new knowledge roles for possibility tables.

Generate **alternative classes and class components** — A column header and its cells represent design components or functions and possible alternatives for the component.

Generate **compatibility constraints** — Constraints between cells within a column or between columns record incompatibilities and help the designer find paths through the table. These appear in the table or on a *compatibility tree* (Figure 22).

Generate a **solution path** — Build a set of links between cells that represent a partial or complete solution class. These may be specified by the designer or through semi-automatic *path generation* based on optimizing solution path criteria such as cost and reliability.

Generate **solution classes** — Cells in the left-most column show names of completed paths through the table. Partial solution paths may suggest new solution classes.

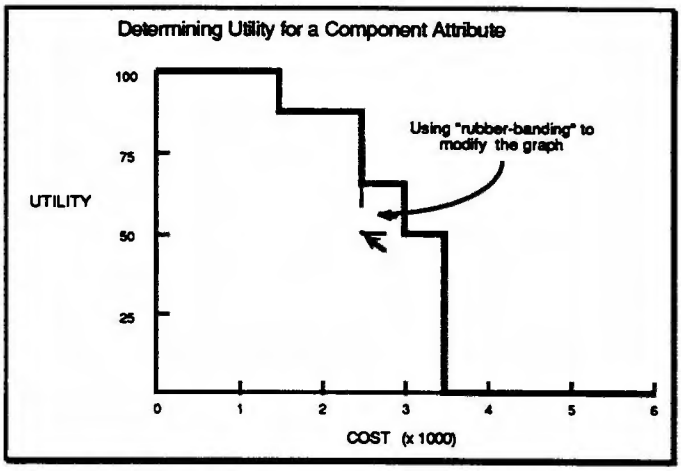


Figure 23. Users map preferences, hard and soft constraints, and values to utility scales.

Generate **solution path criteria** — Global criteria associated with solution paths keep track of items such as cost and reliability. These criteria may reside in an underlying repertory grid and possess weights, preferences, ranges, types, and constraints. A utility mapping tool helps users record preferences and constraints (Figure 23).

Group Use

Experiments using possibility tables in groups are in progress. In the simplest situation, multiple designers contribute non-overlapping information covering different aspects of computer network design. This information combines to form a larger possibility table.

More complex situations will involve:

1) Combining columns and cells in reasonable structural ways
2) Merging different cells under the same column headings
3) Computing differences in constraints when multiple participants share the same cells across columns
4) Measuring content and pointing out conflicts of vocabulary or concepts (similar to the way that Shaw and Gaines measure such differences for repertory grids, above)
5) Specifying constraints between different segments of the model contributed by different participants
6) Analyzing differences in the related repertory grids that hold path criteria
7) Showing consensus and range scores for decisions using this information

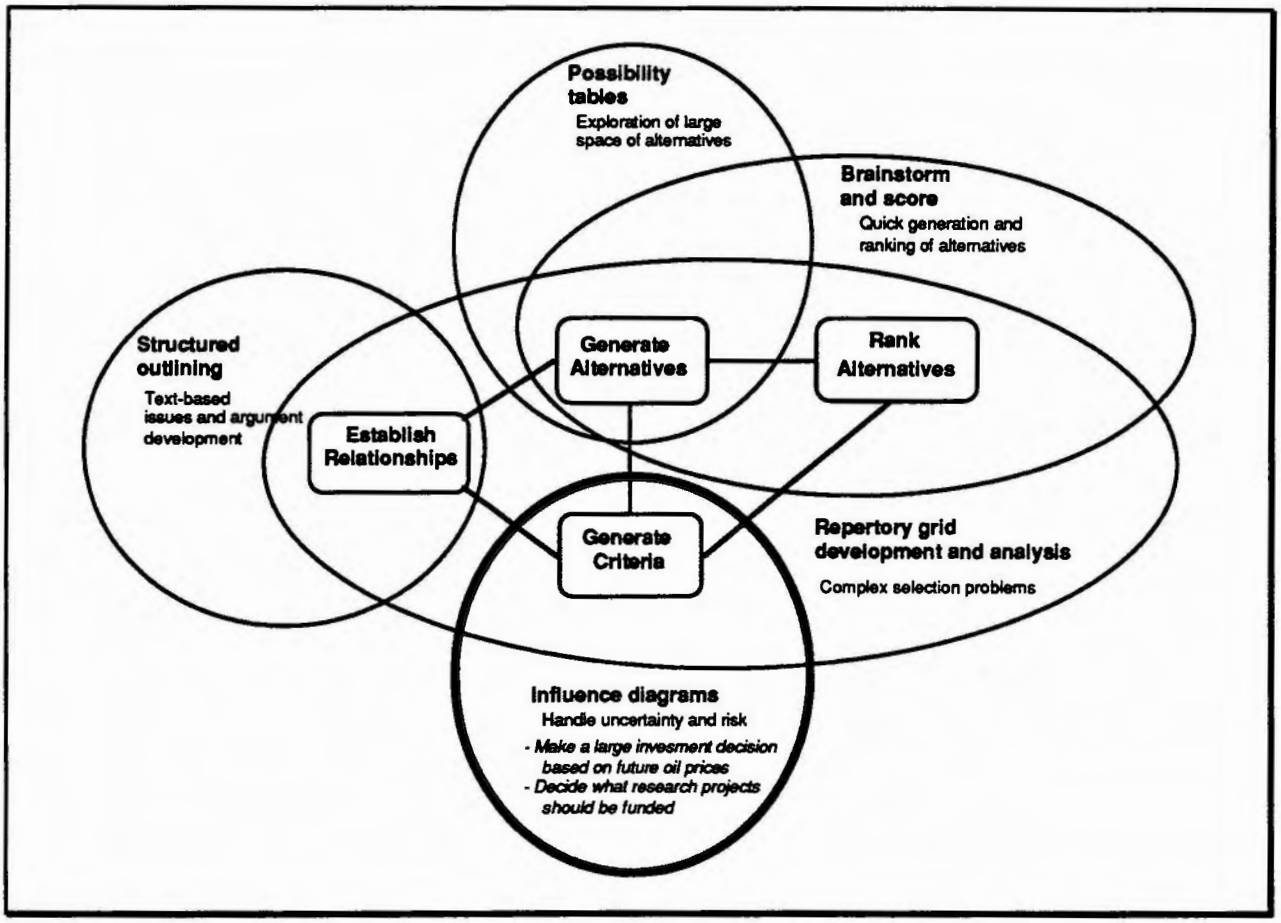


Figure 24. Decision analysis techniques handle conditions of high risk or high uncertainty for some types of important problems. An influence diagrams is a concise form of a probabilistic decision tree showing the conditional relationships between criteria. It enables rigorous decision path scoring. Associated techniques help expand the decision model and uncover areas where it is important to find further information.

We are developing group facilitation processes to resolve differences when they are uncovered. We would like to dynamically measure differences across a group working in parallel on individual yet overlapping parts of a large table.

3.2.5 Part 5: Decision Analysis — Modeling Uncertainty and Risk

Uncertainty and risk are important components of many decisions yet they are seldom included in a formal model (Figure 24). Gathering information about risk and uncertainty may incur a high cost, but automated techniques and good mediating representations can reduce this cost. We have borrowed and extended techniques from decision analysis and knowledge-based systems to build Axotl, a stand-alone decision analyst's workbench. In Axotl, *influence diagrams* from decision analysis (concise form of probabilistic decision trees) show the conditional relationships between criteria (Figure 24). They enable rigorous decision path scoring. Associated techniques can help expand the decision model and uncover areas where it is important to find further information. In the future we will be incorporating methods from Axotl in our group decision support workbench.

Influence diagrams and associated representations address the problems of modeling **enabling conditions, risk, the effects of uncertainty,** and **timing** issues (mentioned in Section 1.3).

Description

A promising approach for dealing with risk and uncertainty is decision analysis (Howard, 1966; Howard and Matheson, 1984; Keeney and Raiffa, 1976; Raiffa, 1968). In the past few years, several tools attempted to help automate portions of the decision analysis process. However, the current generation of automated decision analysis tools (such as decision tree software) are limited in scope and assume a 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 current 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.

Axotl works with knowledge-based templates that contain some of the experience of a decision analyst. It combines a decision analysis workbench with knowledge-based tools to assist individuals consulting with the system about decisions involving high stakes, difficult tradeoffs, or critical uncertainties and risks (Bradshaw and Holtzman, 1987; Bradshaw and Boose 1989; Bradshaw, Covington, Russo, and Boose, 1990, 1991). Axotl contains a graphical editor that

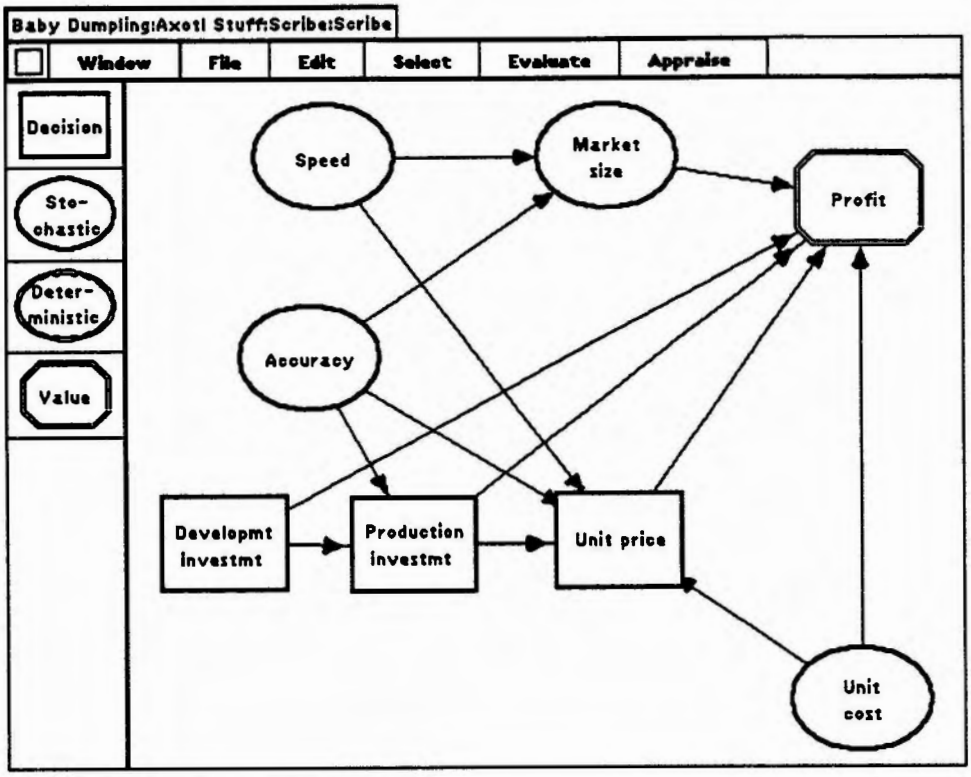


Figure 25. An influence diagram for an R&D investment decision about an automated speech-to-text transcriber. Three decisions form the investment strategy (development investment level, production investment level, and unit price). Rectangular nodes on the diagram represent these decisions. Oval nodes represent technical risk variables (accuracy, speed), production uncertainties (unit cost), and market uncertainties (market size). The eight-sided node labeled "Profit" is the criterion to maximize in evaluating the decision model to determine an optimal policy. Arrows between nodes represent relevance or information flows between variables For example, arrows from "Speed" and "Accuracy" to "Market size" represent judgements about the relevance of technical achievement to our assessment of market size.

helps create and refine influence diagrams that model relevant decision alternatives, preferences, and uncertainties. Influence diagrams are solved to obtain recommended actions in a way that is consistent with probability and utility theory (Howard and Matheson, 1980). The system computes an *expected value* or *utility* for each alternative that expresses the anticipated range of benefit or cost that for a given course of action.

While mathematically similar to probabilistic decision trees, influence diagrams possess several advantages: 1) influence diagrams grow linearly in their graphical representation as contrasted with the exponential growth of trees; 2) they can represent and exploit conditional independence; and 3) in implementation they connect to external procedures and functions in a straightforward way. Additionally, our experience confirms that influence diagrams are an effective way of communicating important issues among participants in a decision, even for those who may not understand the mathematical underpinnings.

The influence diagram solution method implemented in Axotl incorporates a new approach that allows a wide range of questions to be answered directly from the diagram, and preserves the entire underlying joint distribution during solution and inference procedures, rather than just the value lottery and decision policy as is usually done (Shachter, 1986). The distribution editor is a another feature unique to Axotl. It helps users structure conditional probability distributions.

Applications

Decision analysis applies to areas such as business portfolio management, environmental issues, facilities investment and expansion, hurricane seeding, investment strategies, market forecasting, medicine, new product introduction, nuclear plant construction, research and development portfolio management, space program planning, strategic planning, and systems engineering (Howard and Matheson, 1984). At Boeing we have used Axotl to build a prototype for making research and development investment decisions. Other applications under way include process management and bone marrow transplant follow up care.

Developers configure the knowledge-based tools in Axotl with *application-independent* knowledge (i.e., knowledge of decision analysis tools and methods) and *application-specific* knowledge (i.e., knowledge about a particular domain) to provide guidance and help during a decision.

Processes, Methods, Knowledge Roles

This section includes typical processes, methods, and new knowledge roles for decision analysis techniques used by Axotl.

Generate alternatives — Generate alternative choices.

Generate alternative values — Enter the amount of relative worth of each alternative using direct entry or *pairwise comparison.*

Generate criteria (influence diagram variables) — Generate the decision variables for an influence diagram.

Generate constraints — Store constraints in a *compatibility tree* structure.

Generate influence diagrams — Show decision alternatives and their values, criteria and their interrelationships, and associated conditional probabilities.

Weight criteria — Measure the relative importance of criteria using *sensitivity analysis*. Sensitivity analysis enables individuals to determine which variables are the most important determinants of final value (e.g., "Is speed more important than accuracy?"). *Value-of-information* analysis is useful in understanding the importance of resolving uncertainty for specific components of the model (e.g., "How much should I spend to pin down the size of the market?"). *Value-of-control* analysis focuses attention on new alternatives that can increase the ability to bring critical uncertainties under our control (e.g., "Should we control unit cost by acquiring a semiconductor company that manufactures the required chips?"). Graphical facilities supporting value-of-information analysis and value-of-control analysis enable those consulting the system to determine the value of undertaking activities to gather new information or generate new alternatives (Howard and Matheson, 1984).

Generate criteria probabilities (uncertainties) — Estimate the uncertainty associated with criteria values. A *distribution editor* helps structure conditional probability distributions (Figure 26). The system also uses *probability wheel* techniques to encode uncertainty. *Knowledge maps* can help break down complex probability assessments into simpler ones.

Generate value-of-information for criteria — Solve the influence diagram to find the value of reducing uncertainty about criteria values.

Generate value-of-control for criteria — Solve the influence diagram to find the value of being able to control criteria values.

Generate expected values — Solve the influence diagram to find the most reasonable alternative.

Group Use

Merging influence diagram information in group settings will involve:

1) Merging smaller individual influence diagrams into larger, more complex ones
2) Computing and reporting differences in the values of outcomes when participants share the same alternatives
3) Computing and reporting differences in the values of variable probabilities when participants share the same criteria
4) Finding differences in preferences, value, and risk by comparing utility graphs
5) Showing consensus and range scores for expected values of decisions using this information

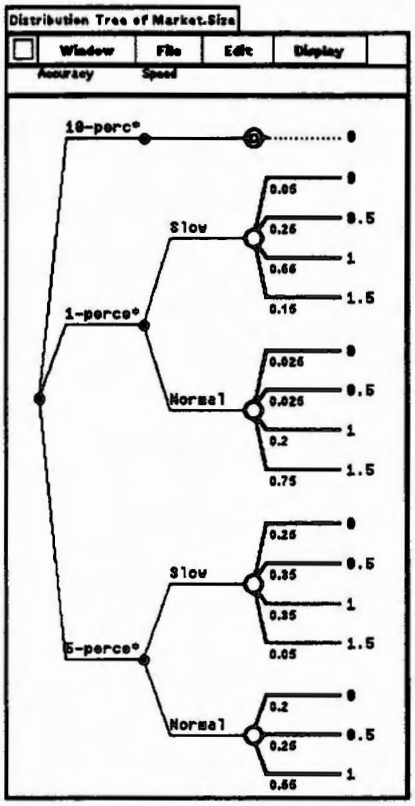


Figure 26. The distribution tree for "Market size". The rightmost nodes and their branches represent the atomic distributions. The atomic distributions of "Market size" are conditioned on its direct predecessors in the influence diagram, "Accuracy" and "Speed". The *conditioning tree* is the part of the distribution tree that precedes the atomic distributions. The ability to explicitly structure conditioning trees and to represent them using a richer language permits reasoning about distributions with additional clarity and efficiency.

4.0 Summary

Existing group decision support systems can help teams reach decisions quickly and efficiently. We pointed out some of the deficiencies of their decision models when using them for complex problems. Weaknesses include the lack of the models' ability to handle documentation of running discussions, complex criteria and alternatives, numeric ranges, exclusivity, enabling conditions, risk management, uncertainty, and timing issues.

We proposed that models used by some successful knowledge acquisition could handle many of these problems. Knowledge acquisition and group decision making are modeling activities. We discussed the importance of good mediating representations. Many of the mediating representations used by knowledge acquisition tools would be useful in a group decision support system. We are in a unique position to combine methods from these areas given our laboratory's extensive experience in implementing and using knowledge acquisition techniques.

Next we pointed out the basic building blocks of decision models — information, preferences, and alternatives. There is a tradeoff between the cost and the benefit of building a more complex model. Techniques such as sensitivity

analysis can help point out when to expand critical parts of the model.

Finally we described components of a complex decision model. We described the modeling strengths contributed by each component. We gave a short description of each component along with the component's methods, applications, knowledge roles (built up from the basic decision element building blocks), and group usage. The components included the brainstorm-and-score model, a gIBIS-like model for recording running discussion, repertory grids for enabling analysis, possibility tables to handle complex alternatives and constraints, and decision analysis techniques to handle risk and uncertainty.

All these components exist in successful tools. Their analyses and methods will be critical to helping groups gain insight and understanding about complex problems. We are blending them into a single workbench that will be powerful enough to handle the complex problems facing Boeing's work teams.

References

Benda, M. (1990). Design of information systems: Towards an engineering discipline, Boeing Computer Services technical report.

Boose, J. H. (1984). Personal Construct Theory and the Transfer of Human Expertise, in the 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*, Nov., 23.

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

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

Boose, J. H. (1989). Using Repertory Grid-Centered Knowledge Acquisition Tools for Decision Support, *Proceedings of the 1989 IEEE Hawaii International Conference on System Sciences*, January.

Boose, J. H. (1991). Knowledge acquisition tools, methods, and mediating representations, in S. Shapiro (Ed.), *Encyclopedia of Artificial Intelligence, 2nd Edition*, Wiley: New York (in press).

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

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 Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 5.1-17.

Boose, J. H., Shema, D. B., Bradshaw, J. M. (1989). Recent Progress in Aquinas: A Knowledge Acquisition Workbench, *Knowledge Acquisition: An International Journal of Knowledge Acquisition for Knowledge-Based Systems*, Vol.1 No.2.

Boose, J. H., Shema, D. B., Bradshaw, J. M. (1990). Capturing Design Knowledge for Engineering Trade Studies, in Wielinga, B., Boose, J., Gaines, B., Schreiber, G., van Someren, M. (Eds.), *Current Trends in Knowledge Acquisition*, IOS Press, Amsterdam.

Boose, J. H., Shema, D. B., Bradshaw, J. M. (1990). Design Knowledge Capture for a Corporate Memory Facility, *Proceedings of the Fifth Conference on Artificial Intelligence for Space Applications*, Huntsville, May.

Bradshaw, J. M., Boose, J. H. (1990). Decision Analysis Techniques for Knowledge Acquisition: Combining Information and Preferences Using Aquinas and Axotl, *International Journal of Man-Machine Studies*, Vol. 32, No. 2, pp. 121-186.

Bradshaw, J. M., Boose, J. H., Covington, S. P., and Russo, P. J. (1989). How to do with grids what people say you can't: The application of decision analysis methods in Axotl and personal construct methods in Aquinas to design problems, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 4.1-15.

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

Bradshaw, J.M., Covington, S., Russo, P., Boose, J.H. (1991). Knowledge acquisition techniques for decision analysis using *Axotl* and *Aquinas*. *Knowledge Acquisition Journal*, in press.

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

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

Chang, E. J. H. (1986). Participant systems, *Future Computing Systems*, 1:3, pp. 253-270.

Chang, E. J. H., (1991). Group coordination in participant systems, *Proceedings of the 24th Annual Hawaii International Conference on System Sciences, Vol. III*, January, pp. 589-599.

Clancey, W.J. (1986). Knowledge Acquisition, presentation at the First Knowledge Acquisiton for Knowledge-Based Systems Workshop, Banff, October.

Clancey, W. J. (1984). Classification problem solving. *Proceedings of the National Conference on Artificial Intelligence*, Austin, Texas.

Clancey, W. (1990). Implications of the system-model-operator metaphor for knowledge acquisition, *Proceedings of the First Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop (JKAW '90)*, Ohmsha Ltd.: Tokyo, pp. 65-80.

Collaboration Technologies Company (1991). VisionQuest Manuals, Collaboration Technologies Company: Austin.

Conklin, J., Begeman, M.L. (1989). gIBIS: A tool for all reasons, *Journal of the American Society for Information Science*, May, pp. 200-213.

Conklin, J., Begeman, M.L. (1988). gIBIS: A hypertext tool for exploratory policy discussion, *ACM*

Transactions on Office Information Systems, 6 (4), pp. 303-331.

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

Cyert, R. M., March, J. G. (1963). *A Behavioral Theory of the Firm*. Englewood Cliffs, New Jersey: Prentice-Hall.

Daniels, R. M., Dennis, A. R., Hayes, G., Nunamaker, J. F. Jr., Valacich, J. (1991). Enterprise Analyzer: Electronic support for group requirements elicitation, *Proceedings of the 24th Annual Hawaii International Conference on System Sciences, Vol. III*, January, pp. 43-52.

Dennis, A. R., George, J. F., Jessup, L. M., Nunamaker, J. F. Jr., Vogel, D. R. (1988). Information technology to support electronic meetings, *MIS Quarterly*, 12:4, pp. 591-624.

Dennis, A. R., Valacich, J. S., Nunamaker, J. F. Jr. (1991). Group, sub-group, and nominal group idea generation in an electronic meeting environment, *Proceedings of the 24th Annual Hawaii International Conference on System Sciences, Vol. III*, January, pp. 573-579.

Dickson, G. W. (1991). An overview of the Minnesota GDSS research project and the SAAM system, in G. R. Wagner (Ed.) *Computer Augmented Teamwork: A Guided Tour*, New York: Van Nostrand Reinhold.

Diederich, J., Ruhman, I., May, M. (1987). KRITON: A knowledge acquisition tool for expert systems. *International Journal of Man-Machine Studies*, 26 (1), pp. 29-40.

Eshelman, L., Ehret, D., McDermott, J., Tan, M. (1987). MOLE: A tenacious knowledge acquisition tool. *International Journal of Man-Machine Studies*, 26, 41-54.

Eshelman, L. (1988). MOLE: A Knowledge Acquisition Tool That Buries Certainty Factors, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, Vol. 29, No. 5, pp. 563-578.

Ford, K.M., Adams-Webber, J.R. (1991). Knowledge Acquisition and Constructivist Epistemology, in *The Psychology of Experts: Cognitive Research and Empirical AI*, Hoffman, R. (ed.). New York: Springer-Verlag (to appear).

Ford, K., Stahl, H., Adams-Webber, J., Novak, J., Jones, J. C. (1990). ICONKAT: An integrated constructivist knowledge acquisition tool, *Proceedings of the 5th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, November.

Gaines, B. R. (1987) Advanced expert system support environments. Boose, J.H. and Gaines, B.R. (Eds) *Proceedings of the Second 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 the 6th International Workshop on Machine Learning*, San Mateo, CA, pp. 156-159.

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

Gaines, B. R., Shaw, M. L. G. (1990). Cognitive and logical foundations of knowledge acquisition, *Proceedings of the 5th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, November.

Garg-Janardan, C., Salvendy, G. (1987). A conceptual framework for knowledge elicitation, *International Journal of Man-Machine Studies*, 26 (4), pp. 521-531.

Gallupe, R. B., DeSanctis, G., Dickson, G. (1988). Computer-based support for group problem finding: An experimental investigation, *MIS Quarterly*, 12:2, pp. 277-296.

Goldratt, E.M. (1990). *Theory of Constraints*. Croton-on-Hudson, NY: North River Press.

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

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

Howard, R. A. (1966). Decision analysis: Applied decision theory. In D. B. Hertz and J. Melese (eds.), *Proceedings of the Fourth International Conference on Operational Research*. Reprinted in Howard and 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.

Jarvenpaa, S. L., Rao, V. S., Huber, G. P. (1988). Computer support for meetings of groups working on unstructured problems: A field experiment, *MIS Quarterly*, 12:4, pp. 645-665.

Johansen, R. (1988). *Groupware: Computer Support for Business Teams*, The Free Press: New York.

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

Kahneman, D., Slovic, P., Tversky, A. (1982). *Judgment Under Uncertainty: Heuristics and Biases*. Cambridge: Cambridge University Press.

Karbach, W., Linster, M., Voß, A. (1990). Model-based approaches: One label - one idea?, in B. Wielinga, J. Boose, B. Gaines, G. Schreiber, and M. van Someren (Eds.), *Current Trends in Knowledge Acquisition*, IOS Press: Amsterdam, Washington, Tokyo, pp. 173-189.

Keeney, R. L. and Raiffa, H. (1976). *Decisions With Multiple Objectives: Preferences and Value Tradeoffs*. New York: Wiley.

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

Klinker, G. (1989). A Framework for Knowledge Acquisition, Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems, Paris, July, 1989, pp.102-116.

Kramer, K.L., King, J.L. (1988). Computer-based systems for cooperative work and group decision making, *ACM Computing Surveys*, Vol. 20, No. 2, pp. 115-146.

Kunz, W., Rittel, H. (1980). APIS: A concept for an argumentative planning information system. Working Paper No. 324, Institute of Urban and Regional Development, University of California at Berkeley.

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

Levi, I. (1986). *Hard Choices: Decision Making Under Unresolved Conflict*. Cambridge: Cambridge University Press.

March, J. G. (1978). Bounded rationality, ambiguity, and the engineering of choice. *The Bell Journal of Economics*, 9(2), 587-608.

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, CA: Scientific Press.

Musen, M. A. (1988). An Editor for the Conceptual Models of Interactive Knowledge-Acquisition Tools, *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 23.1-20.

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

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

Nunamaker, J. F. Jr. (1991). Facilitation in Group Support Technologies, tutorial notes, *24th Annual Hawaii International Conference on System Sciences*, January.

Nunamaker, J. F. Jr., Applegate, L. M., Konsynski, B. R. (1988). Computer-aided deliberation: Model management and group decision support, *Operations Research*, 23:6, pp. 826-848.

Nunamaker, J. F. Jr., Vogel, D., Heminger, A., Martz, B., Grohowski, R., McGoff, C. (1989). Group support systems in practice: Experience at IBM, *Decision Support Systems* 5:2, pp. 183-196.

Nunamaker, J. F. Jr., Weber, E. S., Smith, C. A. P., Chen, M. (1988). Crisis planning systems: Tools for intelligent action, *Proceedings of the 21st Annual Hawaii International Conference on System Sciences, Vol. III*, January, pp. 25-34.

Raiffa, H. (1968). *Decision Analysis: Introductory Lectures On Choices Under Uncertainty*. Reading, Massachusetts: Addison-Wesley.

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

Shachter, R. D. (1986). Evaluating influence diagrams. *Operations Research*, 34: 871-882.

Shaw, M. L. G. (1979). Conversational heuristics for eliciting shared understanding, *International Journal of Man-Machine Studies*, 11, pp. 621-634.

Shaw, M. L. G. (1988). Problems of Validation in a Knowledge Acquisition System Using Multiple Experts, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, pp. 5.1-15.

Shaw, M. L. G., Gaines, B. R. (1987). Techniques for Knowledge Acquisition and Transfer, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 5, *International Journal of Man-Machine Studies*, Vol 27., No. 3, pp. 251-280.

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 Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, pp. 30.1-19.

Shaw, M. L. G., Gaines, B. R. (1989). Comparing conceptual structures: consensus, conflict, correspondence and contrast, *Knowledge Acquisition: An International Journal*, Vol. 1, No. 4, pp. 341-364.

Shaw, M. L. G., Woodward, J. B. (1988). Validation in a Knowledge Support System: Construing Consistency with Multiple Experts, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, Vol. 29, No. 3, pp. 329-350.

Shaw, M. L. G., Woodward, J. B. (1989). Mental Models in the Knowledge Acquisition Process, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 29.1-24.

Shaw, M. L. G., Woodward, J. B. (1990). Modeling expert knowledge, *Knowledge Acquisition*, Vol. 2, No. 3, pp. 179-206.

Shema, D. B., Bradshaw, J. M., Covington, S. P., Boose, J. H. (1990). Design knowledge capture and alternatives generation using possibility tables in Canard, *Knowledge Acquisition*, Vol. 2, No. 4, pp. 345-364.

Sivard, C., Zweben, M., Cannon, D., Laken, F., and Leifer, L. (1989). Conservation of Design Knowledge, *Proceedings of the 27th Aerospace Sciences Meeting (AIAA-89)*, January, Reno, Nevada.

Sycara, K., Roboam, M. (1991). Intelligent information infrastructure for group decision and negotiation support of concurrent engineering, *Proceedings of the 24th Annual Hawaii International Conference on System Sciences, Vol. III*, January, pp. 658-667.

Tan, B. C.-Y., Wei, K.-K., Raman, K. S. (1991). Effects of support and task type on group decision outcome: A study using SAAM, *Proceedings of the 24th Annual Hawaii International Conference on System Sciences, Vol. III*, January, pp. 537-546.

Ventana Corp. (1990). GroupSystems Manuals, Ventana: Tucson.

Wielinga, B., Akkermans, H., Schreiber, G., Balder, J. (1989). A Knowledge Acquisition Perspective On Knowledge-Level Models, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, pp. 36.1-22.

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

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.

Yakemovic, K.C.B., Conklin, E.J. (1990). Report on a development project use of an issue-based information system, Proceedings of the Conference on Computer-Supported Cooperative Work, October, pp. 105-118.

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

Sharable Ontologies as a Basis for Communication and Collaboration in Conceptual Modeling

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

Douglas Skuce, Timothy C. Lethbridge
Department of Computer Science, University of Ottawa
Ottawa, Ontario, Canada K1N 6N5; (613) 564-4518; doug@csi.uottawa.ca

ABSTRACT

We are interested in improving collaboration among researchers, domain experts, and knowledge engineers through better knowledge sharing. Building models is not only a way to *formulate* domain knowledge, but serves more importantly as a means to *communicate* and *come to understand* the evolving problem space. An important prerequisite to effective communication is mutual agreement on important terms and concepts, the *ontology* of the domain. We give an overview of the CODE4 conceptual modeling representation and the DDUCKS modeling environment. We focus on how different classes of ontologies are developed within the tools: top level ontologies with very general concepts, ontologies with modeling concepts, and ontologies for particular domains. We are evaluating theories and contexts as mechanisms for partitioning and managing relationships between groups of concepts. We describe some of the mechanisms we are exploring for exchanging these ontologies with other research groups, and conclude by reviewing important research issues for future work.

1. INTRODUCTION: MODELING AS COMMUNICATION

Knowledge acquisition is largely a matter of communication. It begins when experts in some domain determine that they have valuable information to share. It evolves as participants collaborate to define models that represent a common understanding of certain aspects of the domain. It succeeds when participants can use these models effectively to promote and enrich communication.

This view of knowledge acquisition as modeling is at odds with early knowledge acquisition metaphors that stressed notions of *expertise transfer* and *knowledge acquisition bottlenecks* (Ford & Bradshaw, in press). Clancey (in press) has explained why the bottleneck metaphor is so misleading. He shows that knowledge acquisition is not a matter of squeezing pre-defined chunks of knowledge through a narrow communication channel, but "usually involves inventing new languages for modeling previously unarticulated experience." Clancey's arguments parallel those of Michael Reddy (1979) who criticizes the *conduit metaphor* as a characterization of language and meaning:

> "When one person talks to another, words do not come out of the speaker's mouth as packages full of direct report on facts, which the hearer then opens up to discover the fully packed meaning inside. On the contrary, the speaker forces air out of the lungs and shapes the molecules with the vocal apparatus; these molecules vibrate on the hearer's eardrums, sending neurochemical impulses to the brain. The hearer then *constructs a model* of what the speaker is thought to have said. People understand each other and communicate not by direct conveyance at all, but by broadcasting signals, which themselves evoke mental models." (Frawley, 1992)

Modeling is a *contextual* and *purposive* activity—that is, to be involved in modeling is necessarily to be engaged in using the model in some particular setting for particular reasons that together determine what should be modeled, how to model it, and what can be ignored (Thimbleby, 1990; Winograd & Flores, 1986). Together, the criteria of purpose and cost-effectiveness determine how additional pragmatic issues should be resolved such as who the users of the model are, how it ought to be presented in order to be both usable and useful, and how it will be maintained over its projected lifetime (Rothenberg, 1989).

From a constructivist perspective, a model is not a "picture" of the problem, but rather a device for the attainment or formulation of knowledge about it (Kaplan, 1963). Indeed, sometimes the most important outcome of the modeling process may not be the model itself, but rather the insight we gain as we struggle to articulate, structure, critically evaluate, and rely on it (Moore & Agogino, 1987). Thus, the value of such an effort derives not simply from a final "correct'" representation of the problem, but additionally from our success in framing the activity as a self-correcting enterprise that can subject any part of the model to critical scrutiny, including our background assumptions. We must ask ourselves not only, How do we know the model is correct? (every model is an incorrect oversimplification); but also, How useful is the model (and the modeling process) in facilitating our understanding of the domain?

Hence we see that much of the power of models lies in their ability to function as tools for thought, or *cognitive artifacts* (Norman, 1992). Cognitive artifacts can enhance human representational functions through their ability to maintain, display and operate upon information. They can change the way tasks get done by changing the actions required, and distributing actions across time (precomputation) and people (distributed cognition). Because of the major impact that particular representations chosen can have on problem solving, we ought to design the internal and external form of our models with great care and attention (Chandrasekaran, Narayanan, & Iwasaki, in press; Larkin & Simon, 1987). We also must attend to the transformations of representational form that ought to occur to support different aspects of the modeling process (Gaines, Shaw, & Woodward, in press).

Attention to the use of models as cognitive artifacts enhancing communication and cognition is reflected in the research in the field of conceptual modeling:

> *"Conceptual modeling* is the activity of *formally* describing some aspects of the physical and social world around us *for purposes of understanding and communication.* Such descriptions, often referred to as *conceptual schemata,* require the adoption of a formal notation, a *conceptual model* in our terminology. Conceptual schemata capture relevant aspects of some world, say an office environment and the activities that take place there, and can serve as points of agreement among members of a group, for example the workers in that office, who need to have a common understanding of that world. Conceptual schemata can also be used to communicate that common view to newcomers, through a variety of graphic and linguistic interfaces. Conceptual modeling has an advantage over natural language or diagrammatic notations in that it is based on a formal

> notation which allows one to 'capture the semantics of the application'. It also has an advantage over mathematical or other formal notations developed in computer science because unlike them, conceptual modeling supports structuring and inferential facilities that are psychologically grounded. After all, the descriptions that arise from conceptual modeling activities are intended to be used by *humans*, not machines." (Mylopoulos, 1991)

Unfortunately, until recently developers of modeling and knowledge representation tools have paid scant attention to the needs of the kind of users who are building a model with the intent of coming to understand some domain. For example, traditional data modeling tools introduce assumptions about the way conceptual schemata will be realized on a physical machine or database implementation, which constrain and confound domain experts preferring to leave these considerations to others. Furthermore, the need of such users to model the "real world" drives a requirement for much richer representations than modeling tools typically afford. Knowledge representation tools, on the other hand, almost always introduce the assumption that the resultant knowledge bases will be directly usable for some computational task. Considerations of efficiency and performance inevitably lead to limitations in the richness and flexibility of the knowledge representation. No less important, there are few modeling tools that have been optimized for *usability*. Tools are cumbersome, unnatural, and difficult to use without specialized training and extensive experience. The comprehensibility and succinctness of textual and graphical notations chosen is rarely considered (Bradshaw & Boose, 1992; Chandrasekaran, Narayanan, & Iwasaki, in press). As a result, many potential users abandon modeling tools altogether and use simple graphical drawing tools instead.

Elsewhere we have discussed our ideas about how the application of certain general principles of knowledge representation and user-interface design can make knowledge acquisition tools more usable and extensible by non-experts (Bradshaw, Ford, & Adams-Webber, 1991; Ford, Bradshaw, Adams-Webber, & Agnew, in press; Skuce, in pressb). Section 2 provides an overview of selected aspects of the CODE4 conceptual modeling representation and the DDUCKS modeling environment. We discuss the meaning triangle (Ogden & Richards, 1923) as a way to help us understand how objects in the "real world" relate to objects in the conceptual model and their expression in a particular syntax. We examine how tools and representations for conceptual modeling can satisfy requirements for richness, flexibility, and tailorability.

In Section 3, we discuss preliminary results in our efforts to create sharable specifications for the concepts upon which our applications depend. In this we join with a number of researchers who have argued the benefits for making conceptual commitments explicit in the form of *ontologies*(Gruber, 1991; Gruber, 1992a; Gruber, 1992b; Lenat & Guha, 1990; Neches, Fikes, Finin, Gruber, Patil, Senator, & Swartout, 1991; Neches, Foley, Szekely, Sukaviriya, Luo, Kovacevic, & Hudson, 1992; Skuce & Monarch, 1990):

> "Consider a planning system based on a theory in which plans are composed of 'steps' which form 'sequences' with specific kinds of 'resource dependencies', and that the search for plans is guided by 'ordering heuristics' and 'optimization criteria.' If one wished to use this planning system, one would need to understand what these words mean, and build a knowledge base in which domain-specific knowledge was formulated in terms that the planning *program* also understands."(Gruber, 1992a)

The term *ontology* is borrowed from the philosophical literature where it describes a theory of what exists. Such an account would typically include terms and definitions only for the very basic necessary categories of existence. However, the common usage of ontology in the knowledge sharing and reuse community is as a vocabulary of representational terms and their

definitions at any level of generality. A knowledge-based system's "ontology" defines what exists for the program—in other words, what can be represented by it (Gruber, 1992b).

We focus on how different classes of ontologies are developed within the tools: top level ontologies with very general concepts, ontologies with modeling concepts, and ontologies for particular domains. We discuss theories and contexts as mechanisms for partitioning and managing relationships between groups of concepts. We describe some of the mechanisms we are exploring for exchanging these ontologies with other research groups, and conclude by reviewing important research issues for future work.

2. AN OVERVIEW OF CODE4 AND DDUCKS

2.1. Three-Schema Architectures, Predication, and the Construction of Meaning

There are two general approaches to meaning: one which posits a *direct relation* between symbols and their referents (Haiman, 1985; Wittgenstein, 1974/1918), and another (now dominant) approach which posits an *indirect relation*(Frawley, 1992; Jackendoff, 1983; Lakoff, 1987; Perez, 1992; Sowa, 1984). The roots of the indirect view are usually traced from Aristotle through the more recent *conceptualist position* proposed by Ogden and Richards (Ogden & Richards, 1923). They characterized meaning as a *semiotic triangle* (or *meaning triangle),* a relation between a symbol (term, icon) and a referent (object, extension), mediated by thought (concept, intension, idea, sense) (see Figure 1).[1]

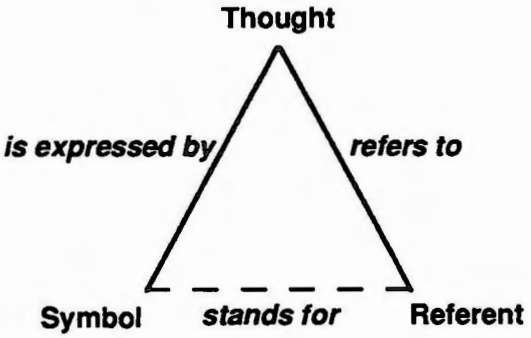


Figure 1. The meaning triangle (Ogden & Richards, 1923).

The meaning triangle is useful for understanding how objects in the "real world" relate to objects in the conceptual model and their expression in a particular syntax (Fulton, 1992). Figure 2 shows a set of interlocking meaning triangles illustrating relationships between employees. We have inverted the triangle so that referents *("real world" objects)* appear near the bottom of the figure, symbols *(the abstract syntax)* in the upper left, and mediating concepts *(the abstract objects constituting an abstract set-theoretic ontology)* in the upper right. Each term in the model *(Tom, Betty)* is linked to a corresponding object in the domain of discourse and a referent in the world. The predicate *works for* expresses the set of all *occurrences* and refers to the three

---

[1] As Sowa (Sowa, 1984) observes, the aspects of meaning described by Ogden and Richards reflect the distinction between *semantic* and *episodic* memory (Tulving, 1972). Semantic memory, which stores universal principles corresponding to dictionary definitions, is related to intensional meaning. Episodic memory, which stores facts about individual things and events corresponding to history and biography, is related to extensional meaning.

situations in the world being described. The sentence *Tom works for Betty,* on the other hand, is related to a single occurrence of the predicate. An abstract set-theoretic *semantics* is defined by the mapping between the abstract objects and the abstract syntax.

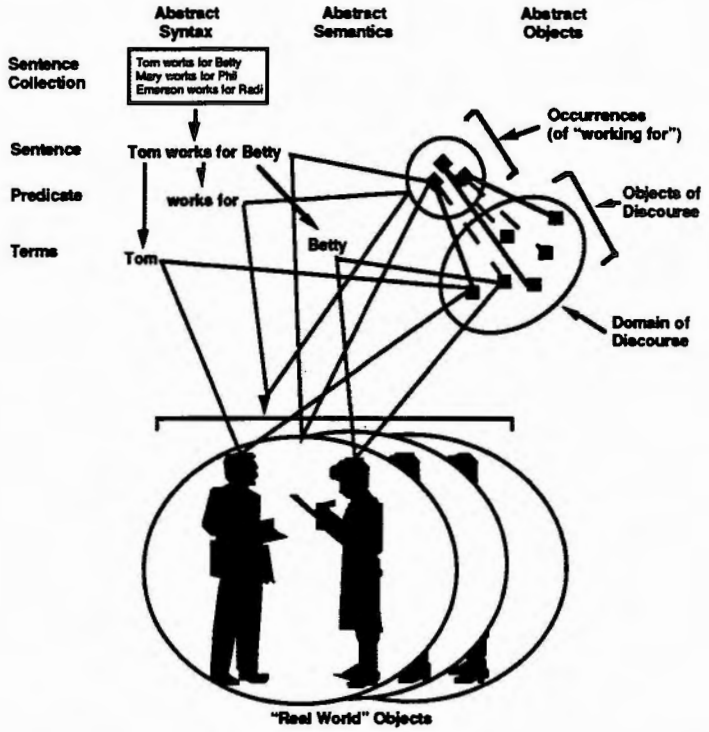


Figure 2. Interlocking meaning triangles show how objects in the "real world" relate to objects in the conceptual model and their expression in a particular syntax (Fulton, 1992).

While a relatively neutral, purely descriptive model of the situation like the one above may be useful for some purposes, it leaves many potential questions unanswered. For example, we may want to get judgmental, *presciptive* advice from a model to help us determine whether Tom *should* be working for Betty or whether we have the optimal number of managers in the organization. To answer these questions, we must construct an interpretation of the situation within a more specialized modeling or problem-solving framework.

Let's say that we are trying to make a decision between Betty, Phil, and Radi as supervisor for Tom. Thus our three options are *Tom works for Betty (d1), Tom works for Phil (d2),* and *Tom works for Radi (d3).* We can represent these options in a variety of ways: for example, as *elements* in a repertory grid, as *alternatives* in an influence diagram or decision tree, or as *possibilities* in a possibility table (Figure 3). While we have argued previously that none of these representations are fundamentally incompatible (Bradshaw & Boose, 1990; Bradshaw, Covington, Russo, & Boose, 1991), they all require different assumptions, vocabulary, and modes of interaction. Within our knowledge acquisition tool, we do not want to commit ourselves to any one of them to the exclusion of the others, because each assists in complementary ways with different aspects of the decision-making process. Furthermore, within The Boeing Company, heterogeneity of tools and notations, and differences in terminology are a fact of life. We have frequently observed the same domain information being used in several ways within different modeling frameworks as data is passed from organization to organization. We need a way to translate between these frameworks without loss of meaning.

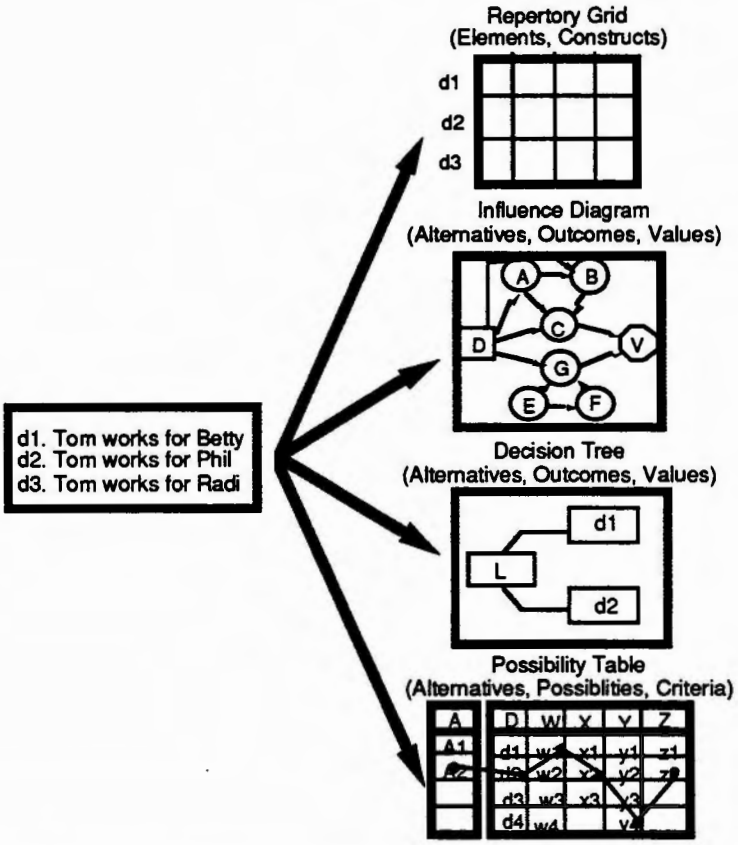


Figure 3. Options for Tom's manager can be represented in different ways.

The problem is one of *semantic unification,* a challenging research topic that is currently being tackled by several research groups (Fulton, Zimmerman, Eirich, Burkhart, Lake, Law, Speyer, & Tyler, 1991; Genesereth & Fikes, 1992; Gruber, 1992a; Gruber, 1992b; Perez, 1992; Sowa & Zachman, 1992). While we do not presume to have an answer to the general problem of semantic unification, we are exploring some relevant issues that bear on the development of our knowledge acquisition tools.

Figure 4 shows a pair of cascading meaning triangles illustrating processes of *abstraction, interpretation,* and *expression* in modeling. In order to facilitate reuse, a relatively neutral domain model is constructed by abstraction based on a mental projection of the real world. The domain model may in turn be interpreted within a more specialized modeling or problem-solving framework, and expressed in terms of a particular visual or textual syntax. The same domain model may be interpreted in multiple ways (e.g., in terms of personal construct theory vs. decision theory); the same interpretation theory may be expressed differently (e.g., as decision trees vs. influence diagrams); and the interpretation model may itself be interpreted within another framework (e.g., a decision-theoretic model interpreted by a domain-specific explanation framework).[2]

---

[2] One way to think about the differentiation between the relatively "neutral" domain model and the interpretation models specialized to different modeling and problem-solving strategies is that it provides a way to address some of

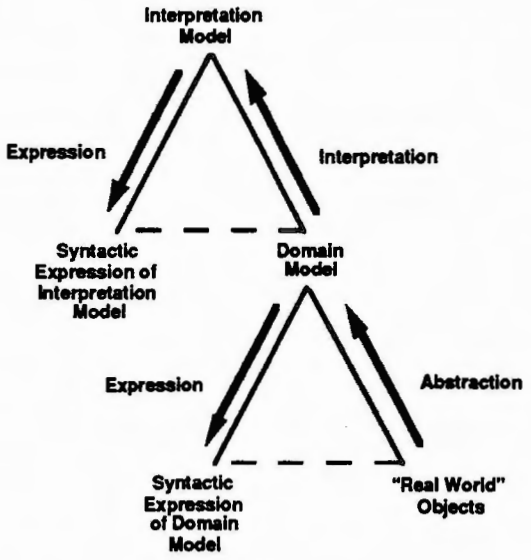


Figure 4. Cascading meaning triangles illustrating processes of *abstraction, interpretation,* and *expression* in modeling.

We discuss now two central problems in building tools to support conceptual modeling:

1. Defining a conceptual modeling representation that is sufficiently rich and flexible (Section 2.2).

2. Defining a capability for users to create mappings between different interpretations and between different expressions of the model (Section 2.3).

2.2. The CODE4 Conceptual Modeling Representation

Ideally, we want a representation that approaches natural language in its ability to describe the subtle details and fluid nature of experience, yet can also be translated straightforwardly into formal, computer-based representations. While we are still so far from the ideal that we do not know even if it will ever be possible to approximate it, a number of significant strides in knowledge representation research have been made over the past few years (Borgida, Brachman, McGuinness, & Resnick, 1989; Gaines, 1991; Genesereth & Fikes, 1992; Neches, Fikes, Finin, Gruber, Patil, Senator, & Swartout, 1991; Perez, 1992; Sowa, 1984; Sowa, 1991). In this section, we briefly describe some of the basic features of the CODE4 conceptual modeling representation, which forms the heart of our approach.

CODE4 is a conceptual modeling tool that has been developed at the University of Ottawa (Lethbridge, 1991; Lethbridge & Skuce, 1992a; Lethbridge & Skuce, 1992b; Skuce, 1991; Skuce, in pressb; Skuce & Lethbridge, submitted for acceptance; Skuce & Monarch, 1990). A collection of integrated tools support the important and frequently overlooked aspects of conceptual, ontological, and terminological analysis. All these tools are available through a rich, graphical interface.

---

the requirements that led (Guha & Lenat, 1990; Lenat & Guha, 1990) to define separate *epistemological* and *heuristic* levels in Cyc.

CODE4 has been used in a number of university and industrial settings as a stand-alone tool. At The Boeing Company, however, we are using the tool in a somewhat different manner. We have developed our own user-interface in the DDUCKS modeling environment, while relying on CODE4 as a back end implementing the conceptual modeling representation. The Boeing group is also developing extensions to the representation to allow the system to make use of additional inferencing and representation facilities similar to those found in Sowa's (Sowa, 1991) conceptual graphs and Gaines' (Gaines, 1991) KRS, which interpret taxonomic and entity-relationship structures in terms of intensional and extensional logics. Currently, any CODE4 representation can be mapped to a conceptual graph, for which semantics exist (Skuce & Lethbridge, submitted for acceptance).

Below, we present a brief summary of some of the basic concepts of the language, as extended in DDUCKS,. See(Lethbridge & Skuce, 1992a; Skuce & Lethbridge, submitted for acceptance) for a more complete discussion of knowledge representation in CODE4.

Things, concepts., types and individuals. Concepts in a CODE4 knowledge base are used to represent things in the world. Things are either types or individuals. Both types and individuals belong to one or more types (the type of "type" is *type*).

Names and unique ID's. Types and individuals have properties and are referred to by terms. The concept's main term is always its name. Concepts always have a term corresponding to a unique ID. The unique ID allows two concepts with the same name to be distinguishable.

Concept hierarchies. Types and individuals are arranged a *kind-of* hierarchy or taxonomy from more general to more specific concepts. Figure 5 shows the hierarchy of basic primitives available in CODE4. Types may have either individuals or more specific types as their subconcepts. Individuals may not have subconcepts.

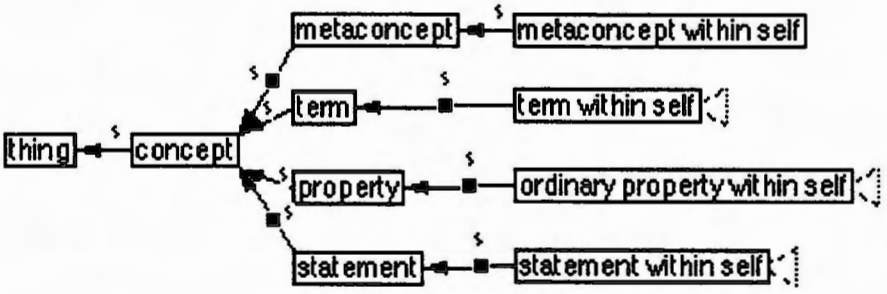


Figure 5. The hierarchy of basic primitives available in CODE4

Inheritance, multiple inheritance and disjointness. Subconcepts typically inherit the properties of their superconcepts. However users have extensive control over the specific aspects of inheritance, if desired. Concepts may inherit from more than one superconcept *(multiple inheritance)*. Sometimes it is impossible for a concept to be a subconcept of a given pair of superconcepts at the same time (e.g., Dick cannot be both a Republican and a Democrat). If this is the case, we specify that the incompatible superconcepts are to be *disjoint*.

Properties and statements. Properties are characteristics that we ascribe to things. We describe things by making statements about them, e.g., "roses are red, violets are blue..."[3]. Statements have at least two essential parts:

- the subject which refers to a thing (roses, violets)
- the predicate which refers to a property of that thing (are red, are blue).

Property hierarchies. Predicates are arranged in a separate hierarchy from things. If a concept has a property, it implies that it also has the superproperty (e.g., *has wheels* implies *has parts*). Every predicate has two parts:

- its most general subject (the most general concept in the concept hierarchy it applies to)
- its implicants (the more general predicates that it implies).

In DDUCKS, the value of a property may be directly asserted by the user or computed by a script.

Attributes and related things. Some important specializations of the property hierarchy are attributes, related things, parts, subsets, state and event predicates, and roles.

Attributes are particular kinds of predicates. User-created attributes will have one of two attribute types as a superproperty: attributes with value or Boolean attributes. Attributes with value take simple values like text, symbols, numbers, bowlines, dates, times, lists, or dictionaries as their values. The system automatically interprets user entered values as the appropriate data type (e.g., *22 June 1990* is interpreted as a Smalltalk object of class *Date*). Values can be specified for a type or an individual. The values of the type are inherited to the individual, which may override them. Boolean attributes are useful when the predicate is naturally unary (e.g., the verb is intransitive (e.g., walks, moves) or the predicate is derived from an adjective (e.g., alive)). Statements about related things take other concepts as their value.

Facets. Facets provide additional descriptive information about statements. Built-in user-editable facets include: status, statement comment, knowledge reference, cardinality, script, legal values, and access parameters. Users can add their own facets.

Metaconcepts and their properties. The system distinguishes between properties ascribed to the thing in the world and the properties of the concept representing the thing. For example:

Concept of Boeing 747 (metaconcept)
    date created: 5 March 1992
    unique ID: @34355
Boeing 747 (concept)
    date created: 1 January 1990
    ID: SN-59874-900

---

[3] *Statements* are roughly equivalent to Fulton's notion of *occurrences*.

Both types and individuals have metaconcepts. The following metaconcept properties are predefined: last modifier, date last modified, creator, date created, description, definition, concept status, unique ID. Most of these are maintained automatically by the system. Additional metaconcept properties can be defined by the user.

Terms. One of the unique strengths of CODE4 is its treatment of linguistic knowledge, especially terminology, i.e., the association between concepts and the phrases that denote them. CODE4 provides a high level of integrated support for lexical functions such as: defining a term, comparing meanings of closely related terms, using a term in several senses, or using synonyms, checking that a term is used consistently and correctly, changing a term throughout a knowledge base, relating verbs to associated nouns (for example, *extract, extractor*), translating a term into another language, and critiquing and assisting in choice of terms. A first order logic system and a simple natural language system allow various types of syntactic and semantic checks to be performed, if desired. A comprehensive lexicon allows references to concepts to be automatically maintained and quickly accessed. We stress the importance of comprehensive lexical support so that terminology can be carefully chosen and subsequently controlled.

2.3. The DDUCKS Modeling Environment

A rich conceptual model can only be managed successfully within a modeling environment that allows users to view and modify the model in forms that are convenient and natural for them (Norman, 1992). The difficulty is that people are language *inventors* more than mere language users. Because the adequacy of a representation in real-world settings is a function of the individuals involved, the situation, the kind of data available, and, most importantly, the questions being asked, people have become adept at back-of-the-envelope innovation, developing new notations on the fly that suit the task at hand. To be effective, conceptual modeling environments must contain a rich set of visual and problem-solving primitives that can be easily adapted and mapped to the conceptual model for different individuals and situations.

Several research groups are working on approaches that allow experts and knowledge engineers to *configure* systems from a set of representational and problem-solving components (Bradshaw, Ford, Adams-Webber, & Boose, in press; Gruber, in press; Marques, Dallemagne, Klinker, McDermott, & Tung, 1992; Marques, Klinker, Dallemagne, Gautier, McDermott, & Tung, 1991; Neches, Fikes, Finin, Gruber, Patil, Senator, & Swartout, 1991; Puerta, Tu, & Musen, in press; Rappaport, 1991). The goal is to create sophisticated object-management and end-user-oriented configuration environments that use increasingly more modular and finely-grained components, thus facilitating radical tailorability, embeddability, and principled reuse. In this section, we provide an overview of elements of the DDUCKS architecture that aim to support these objectives.

At The Boeing Company, we are currently developing problem-solving approaches and ontologies that contain concepts useful in formally describing their work on applications in enterprise modeling and integration (Bradshaw, Holm, Kipersztok, & Nguyen, 1992), design rationale (Bradshaw, Boose, & Shema, in preparation), and group decision support (Boose, Bradshaw, Koszarek, & Shema, 1992). Additionally, Bradshaw and his colleagues are formulating an approach to bone-marrow transplant patient support at the Fred Hutchinson Cancer Research Center (Bradshaw, Chapman, & Sullivan, 1992; Bradshaw, Chapman, Sullivan, Almond, Madigan, Zarley, Gavrin, Nims, & Bush, 1992). The ontologies are being developed within a modeling environment called DDUCKS (Decision and Design Utilities for

Comprehensive Knowledge Support) (Bradshaw, Ford, Adams-Webber, & Boose, in press)[4]. CODE4 has been integrated as a back-end implementation of a conceptual modeling representation for DDUCKS. DDUCKS is based on a three-schema architecture and the client-server model (Bradshaw, Ford, & Adams-Webber, 1991; Ford, Bradshaw, Adams-Webber, & Agnew, in press; van Griethusen & King, 1985). In the future, some portions of the functionality provided by CODE4 will optionally be provided by commercial database products (Figure 6).

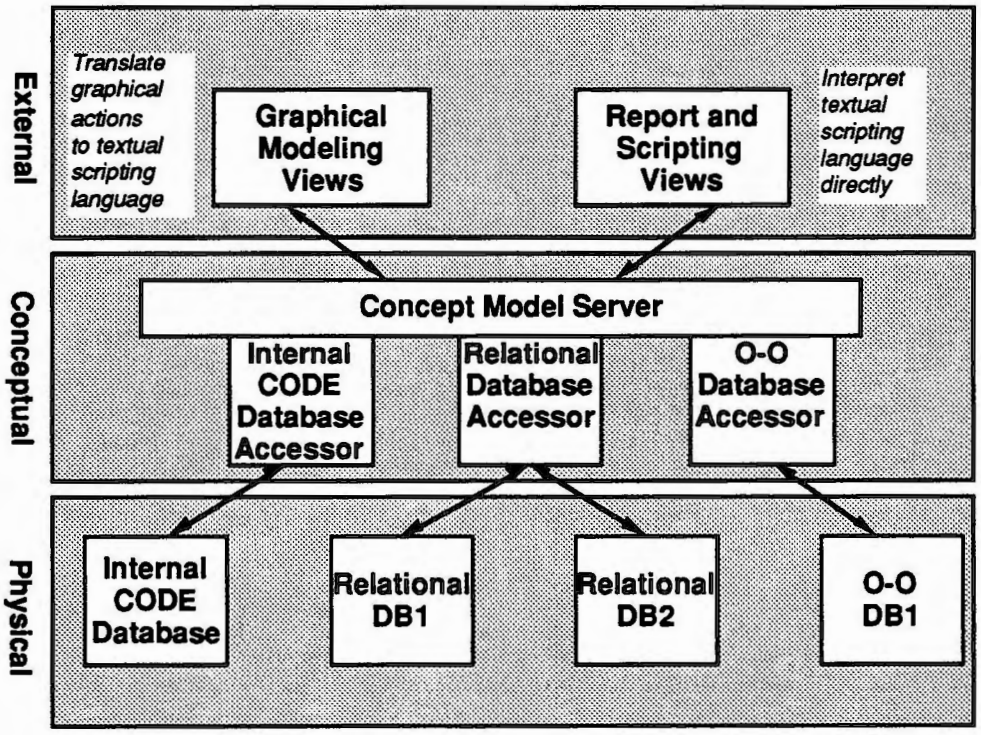


Figure 6. DDUCKS is based on a three-schema architecture and the client-server model.

We describe some of the features that allow DDUCKS to be easily adapted and mapped to the conceptual model for different groups and situations.

The virtual notebook. Individual views containing particular expressions of a model are laid out onto a *page*. Pages are organized within a *virtual notebook,* which helps individuals collect and organize the diverse materials associated with a particular knowledge acquisition project. It also helps manage changes between different versions and views of the model as it evolves. A new notebook is typically opened in double-page mode, displaying a page on the right and one on the left as in a paper notebook. The left page typically contains a table of contents view listing the set of pages available in the notebook. The right page might contain a representation for some portion of the knowledge base. Users move from page to page by selecting a "tab" on the side of the notebook or selecting an item in the table of contents view. Alternatively, the user can query the notebook to bring up pages meeting user-defined criteria. Figure 7 shows a virtual notebook in single-page mode.

---

[4] Pronounced "ducks"

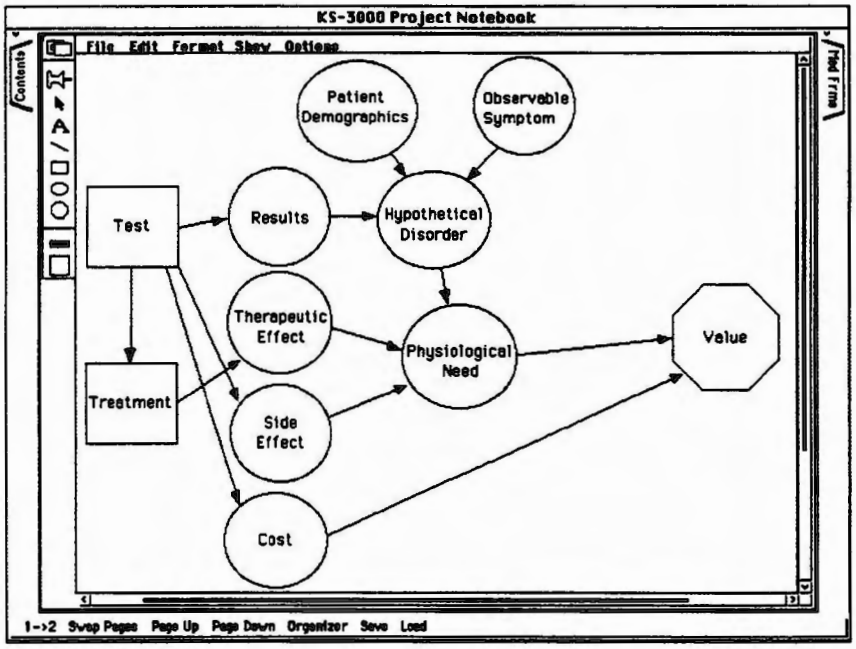


Figure 7. A virtual notebook showing a medical decision-making framing template.

The concept model., scripts, and filters The CODE4 knowledge base in DDUCKS is called the *concept model*. The concept model provides a means semantic unification for information that may be simultaneously portrayed from a number of perspectives in different views (e.g., grids, concept maps, possibility tables, influence diagrams). In DDUCKS, we have defined an integrated scripting and query language so that the environment can be customized by users. Scripts are used for four things: making queries, modifying the structure and behavior of the concept model and views programmatically, defining computed properties, and defining the behavior of complex filters (explained below).

People observe and operate on the concept model *indirectly* by means of syntactic structures implemented in the views. This indirectness can often lead to situations where we are misled in our expectations about the state of the model (Figure 8):

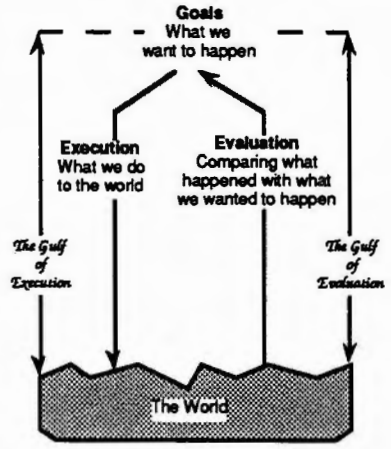


Figure 8. The gulfs of execution and evaluation (Norman, 1992).

> "The gulfs of execution and evaluation refer to the mismatch between our internal goals and expectations and the availability and representation of information about the state of the world [or, in our case, the model] and how it might be changed. The gulf of execution refers to the difficulty of acting upon the [modeling] environment (and how well the artifact supports those actions). The gulf of evaluation refers to the difficulty of assessing the state of the [modeling] environment (and how well the artifact supports the detection and interpretation of that state)...
>
> We can conceptualize the artifact and its interface in this way. A person is a system with an active, internal representation. For an artifact to be usable, the surface representation must correspond to something that is interpretable by the person, and the operations required to modify the information within the artifact must be performable by the user. The interface serves to transform the properties of the artifact's representational system to those that match the properties of the person." (Norman, 1992)

There are two obvious ways to minimize mismatch between the user and the representations embodied in the interface: one is by designing the representations so that they match the user's way of thinking about the model; the other is through mental effort and training on the part of the user to understand the problem in terms of the available representations. Because it is impossible in our setting to assume that users will be willing or able to conform to our representational standard, only the former alternative is really an option.

We attempt to bridge the gulfs of execution and evaluation by providing tools to help users design modeling frameworks and views that fit their personal and organizational requirements. To do this, they must define an explicit specification of how concept model elements, user gestures, and views are mapped to one another (Neches, Foley, Szekely, Sukaviriya, Luo, Kovacevic, & Hudson, 1992). Our goal is to minimize the effort typically required to write procedural code to tailor representations to user needs—we would like creating a new interpretation model and view to require no greater skill in programming than what a user now needs to know to create a macro or chart within a spreadsheet. Our intent is to provide straightforward means to extend a set of generic models declaratively.[5] Since interpretation models and views are represented explicitly as concepts in the concept model, default parameters can be filled-in automatically by inheritance unless a more specialized value is given.

Our general approach in mapping among interpretations and among expressions of the model is similar in many respects to the tool abstraction approach advocated by (Garlan, Kaiser, & Notkin, 1992). The goal is to allow functionality to be enhanced incrementally and modifications to be developed independently in situations where traditional data abstraction approaches are insufficient. Spreadsheets and production systems approximate these objectives: spreadsheets can be enhanced incrementally by adding new equations that operate on existing data; in principle, rules in logic-based production systems can also be developed independently and added to the knowledge base. The essential characteristic of systems that support tool abstraction is that they contain a collection of cooperating *toolies* that provide for mutual notification of changes when shared data structures are modified.

For interpretation model type and each view type, we have defined a set of abstract tools called *filters*. Changes made on a page pass through filters to modify the concept model (Figure 9).

---

[5] Some knowledge of scripting may be still be needed to create more complex interpretation models and views.

Other affected pages are updated through their associated filters. Filter behavior is determined by user-definable scripts, which are executed in response to a change or update request.

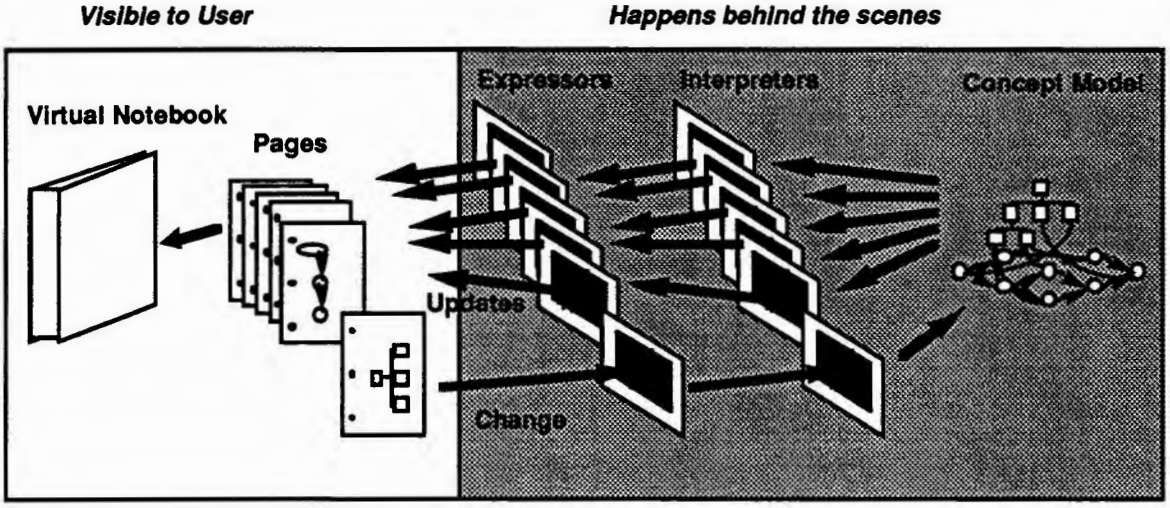


Figure 9. Filters determine how the concept model will be interpreted and expressed in a view. They also determine how actions in a view will be mapped to the concept model.

There are two basic types of filters. *Interpreters* determine which domain concepts are relevant to a view and how those domain concepts (e.g., employee, factory, inspection) will be mapped to interpretation-model-specific concepts (e.g., entity, alternative, activity). *Expressors* determine how and where interpretation model elements are to be expressed as symbolic, syntactical elements within a particular view. Similar kinds of filters determine how actions in the view affect the concept model. New kinds of views can be created on the fly by adding new filters or modifying scripts of existing ones.

3. PRELIMINARY RESULTS IN CREATING SHARABLE ONTOLOGIES

Alexander, Freiling, Shulman, Rehfuss, & Messick (1988) introduced ontological analysis as a knowledge modeling technique for the preliminary analysis of a problem-solving domain. Since that time, several groups of researchers have carried the theme onward, emphasizing the central role of ontologies in facilitating knowledge base reusability (Akkermans, van Harmelen, Schreiber, & Wielinga, in press; Gruber, 1992a; Gruber, 1992b; Lenat & Guha, 1990; Neches, Fikes, Finin, Gruber, Patil, Senator, & Swartout, 1991; Skuce & Monarch, 1990). Well-designed conceptual models can be shared or reused by different tools and applications. For many well-established domains, there is enough similarity between terms and concepts used by various experts that common libraries of ontologies will no doubt flourish. On the other hand, for domains where expert knowledge is highly idiosyncratic, explicit specification of conceptual commitments can promote mutual understanding, even when consensus is neither possible nor desirable.[6]

---

[6] See (Ford & Adams-Webber, 1992) and (Ford & Agnew, 1992) for discussions of the dangers of co-mingling models of idiosyncratic experts within the same knowledge base.

Besides making knowledge sharing easier, careful design of the conceptual terms and definitions helps developers and users of the system. The result of ontological analysis is a rich conceptual model of static, dynamic, and epistemic aspects of the problem. The model can be extended by designers and users of the system and applied to problem-solving. Our experience confirms that terminological confusion breeds conceptual confusion (and vice versa). Such confusion tends to become more probable the higher one goes in a concept hierarchy. For example, it is usually easier to reach consensus on the meaning of the more specialized terms like *pencil* than on general terms like *object*, or *entity*. Hence the most general concepts are the ones we most need to share, yet also the most difficult to agree on.

We anticipate that research on very general ontologies will eventually result in a consensus, although this may be many years away. More specific ontologies use knowledge from more general ontologies, therefore the closer to consensus the higher-level ontologies can be brought, the more effective will be the sharing of lower level ontologies.

At present, the research community is far from agreement as to what the concepts *are*, let alone what to name them. Our approach is therefore three-fold:

1. Develop and evaluate concept specifications for specific domains of interest and problem-solving approaches;

2. Separately, try to develop a set of top-level, general ontological concepts and terminology for future agreement;

3. Evaluate mechanisms to promote knowledge sharing among similarly-minded researchers.

Different classes of ontologies are likely to be maintained by different people. The developers of a tool are likely to be the ones maintaining the specifications for the essential primitives in the conceptual model; modeling and problem-solving methodology experts will maintain the concepts and algorithms that concern them; user groups will define standards for domains and applications to enhance communication among them; and specific users will maintain concepts in the model that describes their particular situation. A layered approach seems well-suited to this kind of tailoring within a modeling environment.

Musen (1989) was one of the first to present an explicit, general approach to creating tailorable knowledge modeling tools. Knowledge modeling tools are tailored using a meta-level tool to edit a domain-independent conceptual model. The meta-level tool, PROTEGE, provides a system to generate knowledge editors tailored for various classes of treatment plans. Physician experts can then use the knowledge editors created by PROTEGE to develop knowledge bases (e.g., OPAL) that encode specific treatment plans in their medical specialty; the resulting systems (e.g., ONCOCIN) could then be used in turn by attending physicians to obtain therapy recommendations for a particular patient. PROTEGE-II generalizes the PROTEGE architecture to allow for alternate problem solving methods and interface styles (Puerta, Tu, & Musen, in press).

Our objective in DDUCKS is to increase reusability by generalizing Musen's approach. It is useful to think of DDUCKS in terms of four "layers" of functionality: workbench, shell,

application, and consultation (Figure 10)[7] . Starting with any layer in the system, a user can produce a set of tools, models, ontologies, and representations that can be used to assist in configuration of a more specialized system at the layer below. See (Bradshaw, Chapman, Sullivan, Almond, Madigan, Zarley, Gavrin, Nims, & Bush, 1992; Bradshaw, Holm, Kipersztok, & Nguyen, 1992) for specific descriptions of the kinds of information besides ontologies that might exist in a given layer for particular applications.

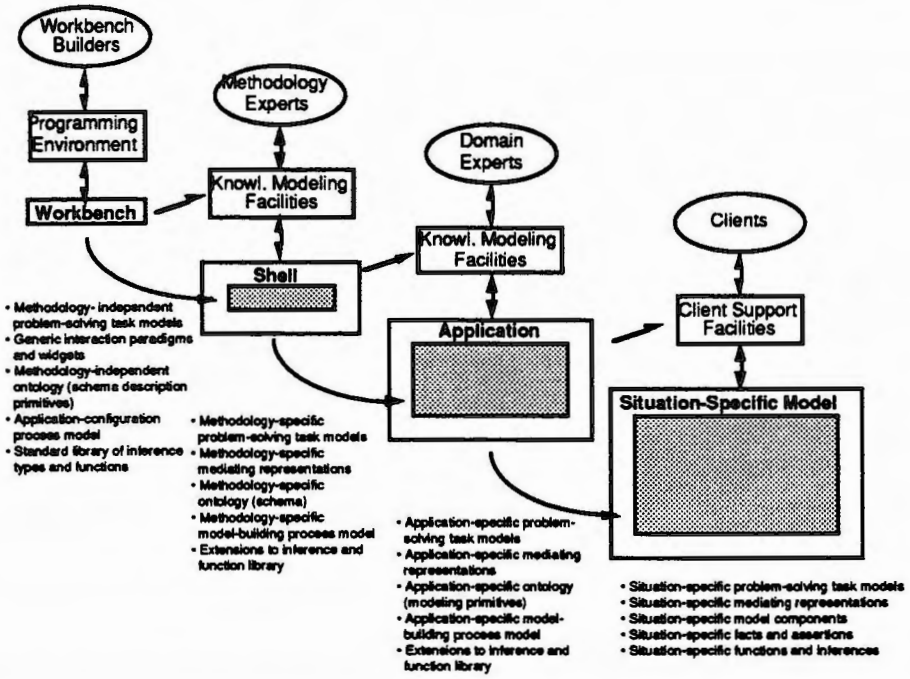


Figure 10. Four layers of functionality facilitate reusability in DDUCKS (inspired by figure from Musen, 1989).

In the following subsection, we discuss how different classes of ontologies are developed within the tools: top level ontologies with very general concepts, ontologies with modeling concepts, and ontologies for particular domains of application (3.1). Then we explain how *theories, contexts,* and *templates* are used to partition and manage groups of concepts (3.2). Finally, we describe how Ontolingua (Gruber, 1992a; Gruber, 1992b) could be used as a mechanism to make ontology sharing possible (3.3).

3.1. Important Kinds of Ontologies in DDUCKS

3.1.1. Very General Ontologies

An important area for of work concerns the most general categories that ontologies can have, notions like: thing, object, entity, property, attribute, event, process, state, situation, collection,

---

[7] The four layers are simply a convenient abstraction that seem to apply to a number of applications. In reality, application configuration and tailoring is a continuous rather than discrete process which admits an unlimited number of "layers".

and relation, to name a few favorites. At the moment, everyone uses these concepts and terms, but: a) they probably use them in very different ways, and b) they cannot tell anyone else what they mean by them. A well known example is the top of the Cyc ontology, which we find difficult to understand from the published descriptions. (Skuce, in pressa).

We believe that such notions can best be clarified by studying linguistic and psychological data. An important ontology based on linguistic research is the Penman ontology (Bateman & Kasper, 1990). This ontology, derived from Halliday's studies of English, is well-documented, with reasonable if minimal descriptions of each of some 200 categories. Other linguistically-based efforts include Miller's WordNet ontology (Miller, 1990), and the semantic analysis of William Frawley (Frawley, 1992). Skuce has been working on an empirical approach to very general ontologies for a number of years. The goal is to prepare an initial proposal with approximately fifty categories of the kind listed in the previous paragraph. The current minimal set of conceptual primitives (e.g., concept, property, term) in CODE4 were described in Section 2.2.

3.1.2. Modeling Ontologies

While the domain ontologies are designed to be relatively neutral with respect to a particular modeling framework, the modeling ontologies are meant to capture a particular theoretical or practical point-of-view. In other words, the modeling ontologies characterize the *roles* that domain concepts play within a particular modeling framework. The knowledge acquisition community has a long and successful history in analyzing knowledge roles as they relate to various problem-solving methods and tasks (Marques, Klinker, Dallemagne, Gautier, McDermott, & Tung, 1991; Puerta, Tu, & Musen, in press). We would like to the analysis of knowledge roles broadened to include not only the abstract function that model elements play computationally in deriving solutions to a problem, but also the roles played in the particular diagrammatic notations describing the model to facilitate communication and *mental* computation (Chandrasekaran, Narayanan, & Iwasaki, in press; Larkin & Simon, 1987). After all, we have argued above that *insight* may well be the most important outcome of a knowledge acquisition project.

Sometimes a modeling methodology embodies both a solution algorithm and one or more particular graphical notations, as decision analysis typically requires either decision trees or influence diagrams (Howard & E., 1984). Sometimes a modeling approach consists of a standard interaction paradigm with an application semantics that is largely user-supplied, as in a spreadsheet. Often at The Boeing Company, people use modeling methodologies that merely define a set of standard drawing conventions for informal diagrams, such as process flow charts.

We are performing an analysis of the knowledge acquisition, decision support, and design rationale tools we have developed over the past several years to understand the concepts and assumptions, implicit and explicit, they embody. We have found that the process of developing a formal modeling ontology for our tools is greatly increasing our understanding of them. As we increase our understanding of the ontological commitments of our own framework, we plan to develop and evaluate similar modeling ontologies derived from other perspectives.

In an application of DDUCKS to airplane design and manufacturing, we have developed an ontology appropriate to describing a large, complex business enterprise. The highest levels of the modeling ontology for conceptual modeling are based largely on the work of (Tauzovich & Skuce, 1990). Within the ontology, a *model* is defined to be the most all-inclusive object of modeling at the highest conceptual level. All other model concepts are either somehow subordinate to the model or external to it. Subtypes of model are used to aggregate the particular types of concepts useful in creating that type of model (e.g., an *enterprise model* representing a

company or university; or a *decision analysis model* comprised of information, preferences, and alternatives). Thus a model type is a definition of the kinds of facts of interest to a certain community for a given purpose, and from which particular descriptions can be built. The union of the description and all concepts external to it is called the *universe of discourse*. The model type determines the rules for determining how appropriate domain model components are to be interpreted before they are optionally expressed in a view.

The use of the term *model* to describe something conveys the idea of a dependency between the thing being modeled and the model of the thing. This dependency relation can be recursive, as when a model of something becomes itself a model for something else. Whether or not something is seen as a model also depends on a particular point of view: When we are viewing something as a model in relation to something else, our concern is how well the model characterizes of the things being modeled and the facts known about them. When this is the concern the language used to build the model is not important as long as each user of the model is able to understand the expressions of that language and, if desired, to translate it without loss of information to the language in which that user would prefer to work (Fulton, Zimmerman, Eirich, Burkhart, Lake, Law, Speyer, & Tyler, 1991).

As a subtype of *concept*, we have defined the root of the hierarchy of primitives of which models are composed: *model concept* (Figure 11). We distinguish two subtypes of model concept: *major model concept* and *minor model concept*. Major model concepts are *model entities, model relationships,* and *model situations;* minor model concepts are *model domains, model properties,* and *model constraints*.

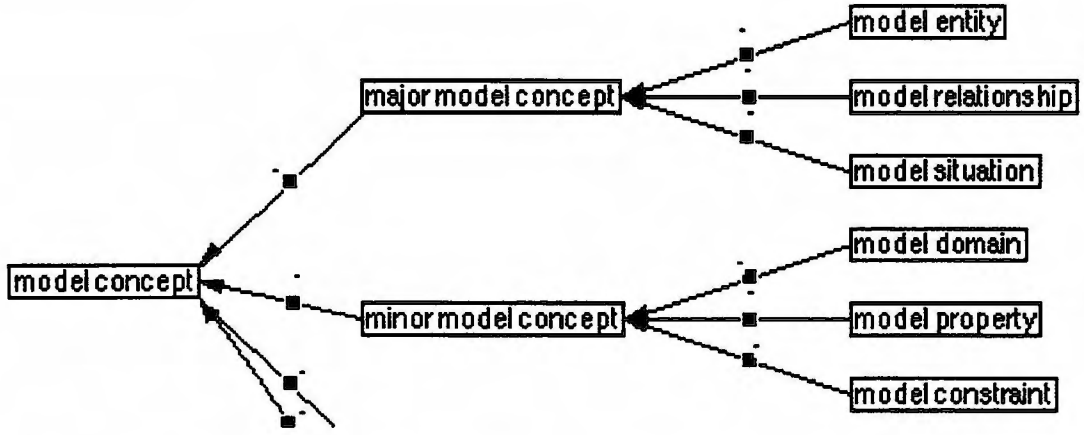


Figure 11. The top of a modeling concept ontology used for an airplane design and manufacturing application..

Model entities are described as follows:

> "An entity is a major concept that represents a class of individual things, persons, places, organizations, projects, etc. which are of relevance to the model. Instances of entities come into existence and cease to exist. An entity has at least one *identifier* (also termed *key property* or *identifying property*) by which instances of the entity are uniquely identified; it may have other properties, some of which may change over time. In natural languages, entities are nearly always named by nouns. The use of the singular for entity names is encouraged." (Tauzovich & Skuce, 1990)

Model entities are typically associated with each other by virtue of the specific roles they play in a common relationship. *Model relationships* are best described as a sort of associative entity:

> "An associative entity [model relationship] is an entity that is introduced to represent a connection between (or among) two or more entities... This connection is being treated as an entity rather than as a relationship, which is done typically to associate or aggregate entities, to eliminate many-to-many relationship, or to allow the connection to have other than solely pre-defined properties." (Tauzovich & Skuce, 1990)

Role properties are used to associate relationships with other concepts. For example, in a *sale* relationship, a role may be defined for a *buyer*, a *seller*, and a *thing being sold*. The particular buyer, seller, and thing sold for that relationship comprise the relationship's *participants*.

Model situations (states of affairs) are used to describe a finite configuration of some aspect of the world (Barwise & Perry, 1983; Sowa, 1984; Sowa, 1992). A situation takes place or exists at some time and location (except for abstract situations, which have no time, location, or causes. Abstract situations are useful in formal subjects, e.g., the notion of environment in a programming language). Situations have a number of participants. Relationships like cause and purpose can be associated with them. They can usually be denoted by a sentence implying a time and a place. A situation may be relatively static, or it may include dynamic processes and events.

Properties describe an aspect of some other concept (see Section 2.2 above). *Domains* are minor concepts that are used to define a set of values for certain properties (e.g., integers from 1-100; {male, female}). *Constraints* are specifications of relations and values that must be maintained.

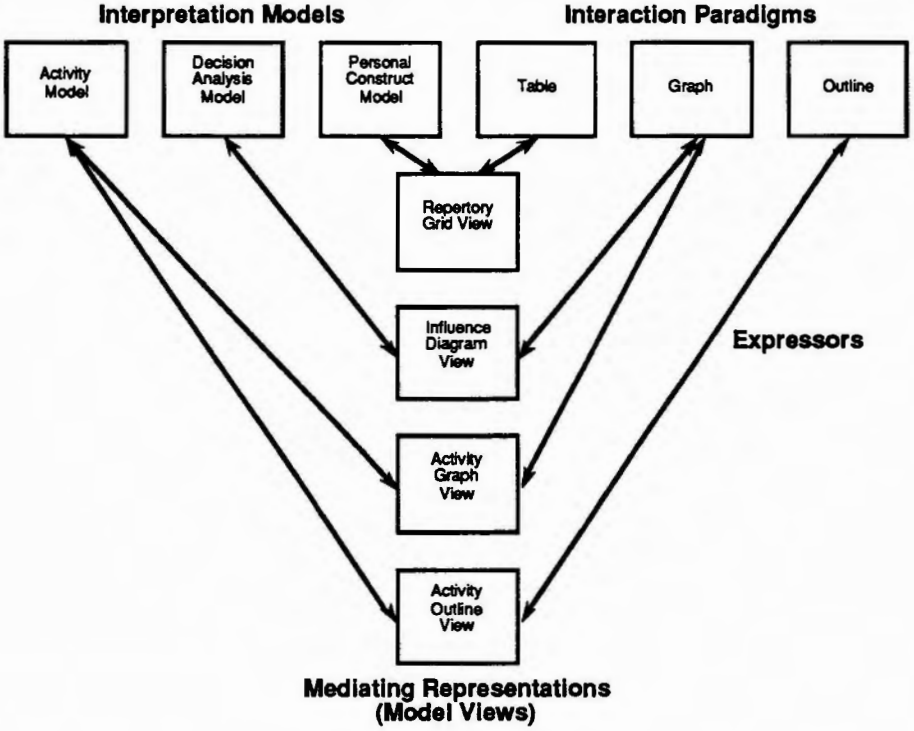


Figure 12. The semantics of a *mediating representation (model view)* are defined by *expressors* that map between a kind of *interpretation model* and particular *interaction paradigm*.

The semantics of a *mediating representation (model view)* are defined by *expressors* that map between a kind of *interpretation model* and particular *interaction paradigm.* At present, concepts that stand for six interaction paradigms have been defined in the ontology (tables, graphs, trees, outlines, lists, and text). The same model type may be expressed by in terms of different interaction paradigms (e.g., *activity models* as *graphs* vs. *outlines*; see Figure 12); conversely, the same interaction paradigm may be reused to create an expression for two different interpretation models (e.g., *graph interaction paradigm* applied to show *influence diagrams* or *activity graphs).* Subtypes of model views defined in the ontology inherit by default the properties and behavior of their supertypes.

3.1.3. Domain Ontologies

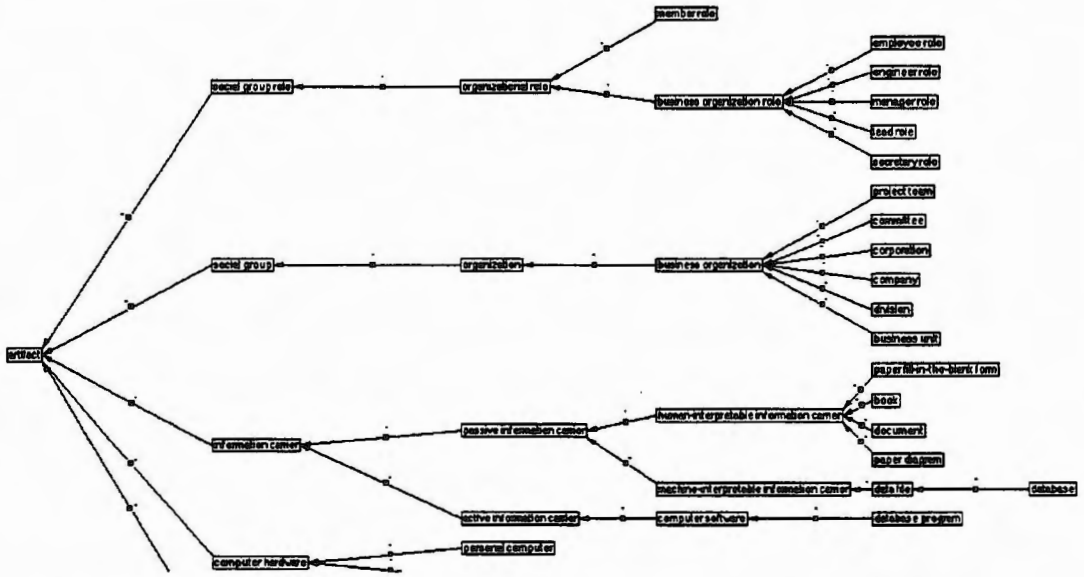


Figure 13. Fragment of a DDUCKS concept model for the airplane design and manufacturing application.

In conjunction with other research groups interested in knowledge sharing issues, we are participating in an effort to develop ontologies for specific domains of interest. For example, at Boeing we have been developing ontological primitives in DDUCKS to represent knowledge of the airplane design and manufacturing enterprise (*eQuality;* see Figure 13) (Bradshaw, Holm, Kipersztok, & Nguyen, 1992). At the Fred Hutchinson Cancer Research Center, we are interested in representing knowledge about bone-marrow transplant follow-up care (Bradshaw, Chapman, & Sullivan, 1992; Bradshaw, Chapman, Sullivan, Almond, Madigan, Zarley, Gavrin, Nims, & Bush, 1992). Others have begun projects that will result in ontologies relevant to disciplines such as process planning, software engineering, circuit layout, and device modeling (Gruber, Tenenbaum, & Weber, 1992).

Domain experts and knowledge engineers jointly define interpretors that map the domain model to an interpretation model. Domain-specific interpretation models and mediating representations may also be defined to fit the needs of specific user groups.

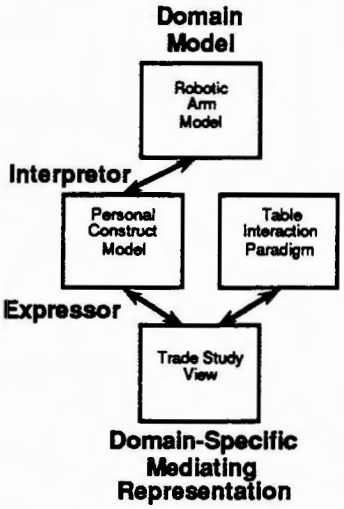


Figure 14. The *domain model* mapped to an interpretation model and expressed in a domain-specific view.

3.2. Theories and Contexts

Practical concerns about the engineering of large systems have led to the development of techniques for creating and operating on multiple partitions of the knowledge base (Gruber, 1992a; Guha, 1991). Each partition, or *theory*, constitutes a set of axioms tailored for a particular domain or purpose. Each theory is associated with a *context* (McCarthy, 1987) that describes, among other things, the set of assumptions made by the theory.

Within DDUCKS, groups of concepts are organized into theories. Each concept must belong to one or more theories. Theories are semi-permeable partitions of the concept model. Theories themselves are concepts, which allows for relationships to be defined between a theory and other concepts. Individual theories are organized into hierarchies of supertheories and subtheories. By default, a subtheory imports all concepts of its supertheories. Theories can optionally "import" (i.e., "lift") and "export" concepts selectively.

3.3. A Mechanism for Sharing

The rate of progress on ontological issues in the research community will belargely determined by how well knowledge can be shared. Ontological research is currently shared very little, and where sharing takes place, it is usually either a) in the form of paper reports that take time to distribute and get outdated rapidly, or b) among researchers using some specific piece of not-widely-distributed research software. To facilitate development of ontologies it will be necessary for determine how divers software tools can be facilitated to exchange their knowledge. It will also be necessary to determine *conceptual* formats, i.e. exactly what pieces of information are necessary to convey to accurately communicate ontological concepts among a community of researchers.

Gruber's work on Ontolingua (Gruber, 1992a; Gruber, 1992b) currently provides the most promising mechanism for sharing ontologies between different tools and formalisms. Ontolingua extends the knowledge interchange format (KIF; (Genesereth & Fikes, 1992)) defined by the DARPA knowledge sharing effort with standard primitives for defining classes and relationships,

and organizing knowledge in object-centered hierarchies with inheritance. Ontolingua facilitates the translation of KIF-level sentences to and from forms that can be used by various knowledge representation systems (currently LOOM, Epikit, Algernon, and a canonical form of KIF). We are working with Gruber to define an Ontolingua interface for CKB, the CODE4 knowledge base file format (Lethbridge & Skuce, 1992a).

The goal is to use the Ontolingua project as a basis for processing and distributing the ontologies. As good examples become available, the Sharing and Reuse of Knowledge Bases subgroup of the knowledge sharing effort can coordinate evaluations and experiments. In this way, we can pool our results with those of projects with similar goals.

4. CONCLUSION

In order to facilitate sharing of ontologies a number of key research issues must be addressed.

- Knowledge primitives: *Of what primitive elements is the knowledge composed?* We take the approach that all elements are to be considered as concepts· and consistent with the CODE4 system we categorize concepts. into types, instances, statements, properties, terms, metaconcepts, etc. Simply saying that the primitives shall be some set of "predicates" does not answer the difficult question of where we get this set from. This is where linguistic and terminological research becomes especially relevant.

- Standardization: *Should there be only one format or more than one?* The types of knowledge in ontologies developed within the research community may have special properties that differ significantly from knowledge exchanged in other communities. Some may consider these differences mandate different interchange standards. We believe that the differences are not great enough, and the benefits of a single standard are of over-riding importance. (what about IRDS, SUMM?)

- Inference primitives: *What are the basic semantics of the exchanged knowledge?* For example, in CKB format some basic rules of inheritance are assumed. If somebody attempted to process a CKB knowledge base but assumed different or no inheritance rules, the results of the processing would likely be invalid (or at the very least, substantial knowledge would be lost). We believe it is necessary to have an absolute minimum number of inference primitives otherwise each system will be forced to reimplement every other system. In order to allow the exchange of knowledge between systems that have differing inference primitives, we believe it is important to be able to exchange *informal* knowledge. Such knowledge is only manipulated by systems that recognize it. Informal knowledge can also facilitate human understanding by allowing such things as comments. (analogy EPSF?)

- Physical format: Obviously it will be necessary to develop a common syntax, but there are a number of meta-issues upon which to be agreed: Should compactness be a priority? What about human readability? Should the structure be hierarchical or flat? Should it look like predicate logic or some extension thereto such as conceptual graphs (Sowa, 1984)? How are cross-references represented: by unifiable symbols (English words), by indexes, or by pointers?

We have chosen our own positions on the above issues, and so have others. Now the issues have been raised we anticipate a lively debate. However, we hope existing formats for sharing knowledge will begin to accelerate the exchange of ontologies. (grounded in real exchange, real problems)

We hope that our initial efforts at developing ontologies and sharing mechanisms will spur other researchers to make similar efforts, with the goal of increasing knowledge sharing and reuse within the research community.

ACKNOWLEDGMENTS

We express our appreciation to Russell Almond, Miroslav Benda, Guy Boy, Kathleen Bradshaw, Jim Fulton, Cindy Holm, Earl Hunt, Oscar and Sharon Kipersztok, Cathy Kitto, David Madigan, Allen Matsumoto, Thom Nguyen, Steve Poltrock, Bob Schneble, Doug Schuler, Kish Sharma, Dave Shema, Bruce Wilson, and Debra Zarley for their contributions and support. This work has benefited from discussions with numerous colleagues in the knowledge acquisition community over the years, in particular Ken Ford, Brian Gaines, Tom Gruber, Mark Musen, Mildred Shaw, John Sowa, and Brian Woodward.

REFERENCES

Akkermans, H., van Harmelen, F., Schreiber, G., & Wielinga, B. (in press). A formalisation of knowledge-level models for knowledge acquisition. In K. M. Ford & J. M. Bradshaw (Eds.), Knowledge Acquisition as a Modeling Activity New York: John Wiley.

Alexander, J. H., Freiling, M. J., Shulman, S. J., Rehfuss, S., & Messick, S. L. (1988). Ontological analysis: An ongoing experiment. In J. H. Boose & B. R. Gaines (Eds.), Knowledge Acquisition Tools for Expert Systems (pp. 25-37). London: Academic Press.

Barwise, J., & Perry, J. (1983). Situations and Attitudes, Cambridge, MA: MIT Press.

Bateman, J., & Kasper, R. (1990). A General Organization of Knowledge for Natural Language Processing: The Penman Upper Model, Marina del Rey, CA: USC/Information Sciences Institute.

Boose, J. H., Bradshaw, J. M., Koszarek, J. L., & Shema, D. B. (1992). Better group decisions: Using knowledge acquisition techniques to build richer decision models. *Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW-92)*, . Banff, Alberta, Canada:

Borgida, A., Brachman, R. J., McGuinness, D. L., & Resnick, L. A. (1989). CLASSIC: A structural data model for objects. J. Clifford, B. Lindsay, & D. Maier (Ed.), *1989 ACM SIGMOD International Conference on the Management of Data*, . ACM Press: New York.

Bradshaw, J. M., & Boose, J. H. (1990). Decision analysis techniques for knowledge acquisition: Combining information and preferences using *Aquinas* and *Axotl*. In J. H. Boose & B. R. Gaines (Eds.), Progress in Knowledge Acquisition for Knowledge-Based Systems London: Academic Press.

Bradshaw, J. M., & Boose, J. H. (1992). Mediating representations for knowledge acquisition No. Seattle Washington:Boeing Computer Services.

Bradshaw, J. M., Boose, J. H., & Shema, D. B. (in preparation). A knowledge acquisition approach to design rationale. In J. Carroll & T. Moran (Eds.), Design Rationale Hillsdale, N.J.: Lawrence Erlbaum.

Bradshaw, J. M., Chapman, C. R., & Sullivan, K. M. (1992). An application of DDUCKS to bone-marrow transplant patient support. *Working Notes of the AAAI 1992 Artificial Intelligence in Medicine Session of the Spring Symposium*, . Stanford, California:

Bradshaw, J. M., Chapman, C. R., Sullivan, K. M., Almond, R. G., Madigan, D., Zarley, D., Gavrin, J., Nims, J., & Bush, N. (1992). KS-3000: An application of DDUCKS to bone-marrow transplant patient support. *Submitted to the Sixth Annual Florida AI Research Symposium (FLAIRS '93)*, . Ft. Lauderdale, FL:

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

Bradshaw, J. M., Ford, K. M., & Adams-Webber, J. R. (1991). Knowledge representation for knowledge acquisition: A three-schemata approach. *Proceedings of the Sixth Knowledge Acquisition for Knowledge-Based Systems Workshop*, . Banff, Alberta, Canada:

Bradshaw, J. M., Ford, K. M., Adams-Webber, J. R., & Boose, J. H. (in press). Beyond the repertory grid: New approaches to constructivist knowledge acquisition tool development. In K. M. Ford & J. M. Bradshaw (Eds.), Knowledge Acquisition as a Modeling Activity New York: John Wiley.

Bradshaw, J. M., Holm, P., Kipersztok, O., & Nguyen, T. (1992). *eQuality:* An application of *Axotl II* to process management. In T. Wetter, K.-D. Althoff, J. H. Boose, B. R. Gaines, M. Linster, & F. Schmalhofer (Eds.), Current Developments in Knowledge Acquisition: EKAW-92 Berlin/Heidelberg: Springer-Verlag.

Chandrasekaran, B., Narayanan, N. H., & Iwasaki, Y. (in press). Reasoning with diagrammatic representations: A report on the AAAI Spring Symposium, March 25-27, 1992. *AI Magazine*.

Clancey, W. J. (in press). The knowledge level reinterpreted: Modeling socio-technical systems. In K. M. Ford & J. M. Bradshaw (Eds.), Knowledge Acquisition as a Modeling Activity New York: John Wiley.

Ford, K. M., & Bradshaw, J. M. (in press). Knowledge Acquisition as Modeling, New York: John Wiley.

Ford, K. M., Bradshaw, J. M., Adams-Webber, J. R., & Agnew, N. M. (in press). Knowledge acquisition as a constructive modeling activity. In K. M. Ford & J. M. Bradshaw (Eds.), Knowledge Acquisition as a Modeling Activity New York: John Wiley.

Frawley, W. (1992). Linguistic Semantics, Hillsdale, N.J.: Lawrence Erlbaum.

Fulton, J. A. (1992). Semantic Unification and the Meaning Triangle No. Boeing Computer Services.

Fulton, J. A., Zimmerman, J., Eirich, P., Burkhart, R., Lake, G. F., Law, M. H., Speyer, B., & Tyler, J. (1991). The Semantic Unification Meta-Model: Technical Approach No. Dictionary/Methodology Committee of the IGES/PDES Organization, ISO TC184/SC4.

Gaines, B. R. (1991). Empirical investigation of knowledge representation servers: Design issues and applications experience with KRS. *SIGART Bulletin*, 2(3), 45-56.

Gaines, B. R., Shaw, M. L. G., & Woodward, J. B. (in press). Modeling as a framework for knowledge acquisition methodologies and tools. In K. M. Ford & J. M. Bradshaw (Eds.), Knowledge Acquisition as a Modeling Activity New York: John Wiley.

Garlan, D., Kaiser, G. E., & Notkin, D. (1992). Using tool abstraction to compose systems. Computer, June, p. 30-38.

Genesereth, M. R., & Fikes, R. (1992). Knowledge Interchange Format Version 3.0 Reference Manual No. Logic Group Report, Logic-92-1). Stanford University Department of Computer Science.

Gruber, T. R. (1991). The role of common ontology in achieving sharable, reusable knowledge bases. In J. A. Allen, R. Fikes, & E. Sandewall (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference (pp. 601-602). San Mateo, CA: Morgan Kaufmann.

Gruber, T. R. (1992a). Ontolingua: A mechanism to support portable ontologies, Version 3.0 No. Stanford Knowledge Systems Laboratory Technical Report KSL 91-66). Stanford University Department of Computer Science.

Gruber, T. R. (1992b). A translation approach to portable ontology specifications. *Proceedings of the Seventh Knowledge Acquisition for Knowledge-Based Systems Workshop,* . Banff, Alberta, Canada:

Gruber, T. R. (in press). Model formulation as a problem solving task: Computer-assisted engineering modeling. In K. M. Ford & J. M. Bradshaw (Eds.), Knowledge Acquisition as a Modeling Activity New York: John Wiley.

Gruber, T. R., Tenenbaum, J. M., & Weber, J. C. (1992). Toward a knowledge medium for collaborative product development. J. S. Gero (Ed.), *Proceedings of the Second International Conference on Artificial Intelligence in Design,* . Pittsburgh, PA:

Guha, R. V. (1991). Contexts: A formalization and some applications No. Technical Report Number ACT-CYC-423-91). MCC.

Guha, R. V., & Lenat, D. B. (1990). Cyc: A midterm report. AI Magazine, p. 33-58.

Haiman, J. (1985). Natural Syntax, Cambridge: Cambridge University Press.

Howard, R. A., & E., M. J. (1984). Influence diagrams. In R. A. Howard & J. E. Matheson (Eds.), Readings on the Principles and Applications of Decision Analysis Menlo Park, California: Strategic Decisions Group.

Jackendoff, R. (1983). Semantics and Cognition, Cambridge, MA: MIT Press.

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

Lakoff, G. (1987). Women, Fire, and Dangerous Things: What Categories Reveal About the Mind, Chicago: University of Chicago Press.

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

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

Lethbridge, T. C. (1991). Creative knowledge acquisition: An analysis. *Proceedings of the 1991 Banff Knowledge Acquisition for Knowledge-Based Systems Workshop,* . Banff, Canada:

Lethbridge, T. C., & Skuce, D. (1992a). Informality in knowledge exchange. *Working Notes of the AAAI-92 Knowledge Representation Aspects of Knowledge Acquisition Workshop*, 92.

Lethbridge, T. C., & Skuce, D. (1992b). Integrating techniques for conceptual modeling. *Proceedings of the Seventh Annual Knowledge Acquisition for Knowledge-Based Systems Workshop,* . Banff, Alberta, Canada:

Marques, D., Dallemagne, G., Klinker, G., McDermott, J., & Tung, D. (1992). Easy programming: Empowering people to build their own applications. *IEEE Expert*, June, 16-29.

Marques, D., Klinker, G., Dallemagne, G., Gautier, P., McDermott, J., & Tung, D. (1991). More data on usable and reusable programming constructs. *Proceedings of the Sixth Banff Knowledge Acquisition for Knowledge-Based Systems Workshop,* (pp. 6-11). Banff, Canada:

McCarthy, J. (1987). Generality in artificial intelligence. *Communications of the ACM*, 30(12), 1030-1035.

Miller, G. (1990). WordNet: an on-line lexical database. *International Journal of Lexicography*, 3(4).

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. (1989). Automated Generation of Model-Based Knowledge-Acquisition Tools, San Mateo, CA: Morgan Kaufmann.

Mylopoulos, J. (1991). Conceptual modeling and Telos No. Technical Report DKBS-TR-91-3). University of Toronto Department of Computer Science.

Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T., & Swartout, W. R. (1991). Enabling technology for knowledge sharing. AI Magazine, p. 36-55.

Neches, R., Foley, J., Szekely, P., Sukaviriya, P., Luo, P., Kovacevic, S., & Hudson, S. (1992). Knowledgeable development environments using shared design models No. USC/Information Sciences Institute and Georgia Institute of Technology.

Norman, D. A. (1992). Cognitive artifacts. In J. M. Carroll (Eds.), Designing Interaction: Psychology at the Human-Computer Interface (pp. 17-38). Cambridge: Cambridge University Press.

Ogden, C. K., & Richards, I. A. (1923). The Meaning of Meaning, New York: Harcourt, Brace, and Company.

Perez, S. (1992). Model unification for data repositories No. Technical Report Number X3H4.6/92-001, Version 0.2). American National Standards Institute (ANSI).

Puerta, A., Tu, S., & Musen, M. (in press). Modeling tasks with mechanisms. In K. Ford & J. M. Bradshaw (Eds.), Knowledge Acquisition as a Modeling Activity New York: John Wiley.

Rappaport, A. (1991). A theory of local and interstitial knowledge. *Proceedings of the Sixth Knowledge Acquisition for Knowledge-Based Systems Workshop*, . Banff, Alberta, Canada:

Reddy, M. (1979). The conduit metaphor: A case of frame conflict in our language about language. In A. Ortony (Eds.), Metaphor and Thought Cambridge: Cambridge University Press.

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

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

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

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

Skuce, D., & Lethbridge, T. C. (submitted for acceptance). A knowledge representation for interactive knowledge management. *Proceedings of the Third International Conference on the Principles of Knowledge Representation and Reasoning*, . Cambridge, MA:

Skuce, D., & Monarch, I. (1990). Ontological issues in knowledge base design: Some problems and suggestions. *Proceedings of the Fifth Knowledge Acquisition for Knowledge-Based Systems Workshop*, . Banff, Alberta, Canada:

Sowa, J. F. (1984). Conceptual Structures: Information Processing in Mind and Machine, Reading, MA: Addison-Wesley.

Sowa, J. F. (1991). Towards the expressive power of natural languages. In J. F. Sowa (Eds.), Principles of Semantic Networks (pp. 157-189). San Mateo, CA: Morgan Kaufmann.

Sowa, J. F. (1992). Representing and reasoning about contexts. In S. C. Shapiro, J. Barnden, D. Kumar, J. P. Martins, & J. F. Sowa (Ed.), Working Notes for the Propositional Knowledge Representation Symposium of the AAAI Spring Symposium pp. 133-142). Stanford, CA: Stanford University.

Sowa, J. F., & Zachman, J. A. (1992). Extending and formalizing the framework for information systems architecture. *IBM Systems Journal*, 31(3), 590-616.

Tauzovich, B., & Skuce, D. (1990). A general taxonomy for conceptual data modeling No. Cognos.

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

van Griethusen, J. J., & King, M. H. (1985). Assessment guidelines for conceptual schema language proposals No. ISO-TC97/SC21/WG5-3). International Standards Organization.

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

Wittgenstein, L. (1974/1918). Tractatus logico-philosophicus, Atlantic Highlands, N.J.: Humanities Press.

Knowledge Acquisition with a Knowledge-Intensive Machine Learning System

Clifford A. Brunk and Michael J. Pazzani
Department of Information and Computer Science
University of California
Irvine, CA 92717
(brunk@ics.uci.edu)
(pazzani@ics.uci.edu)

Abstract

In this paper, we investigate the integration of knowledge acquisition and machine learning techniques. We argue that existing machine learning techniques can be made more useful as knowledge acquisition tools by allowing the expert to have greater control over and interaction with the learning process. We describe a number of extensions to FOCL (a multistrategy Horn-clause learning program) that have greatly enhanced its power as a knowledge acquisition tool, paying particular attention to the utility of maintaining a connection between a rule and the set of examples explained by the rule. The objective of this research is to make the modification of a domain theory analogous to the use of a spread sheet. A prototype knowledge acquisition tool, FOCL-1-2-3, has been constructed in order to evaluate the strengths and weaknesses of this approach.

1.0 Introduction

The emphasis of our research has been on the integration of machine learning and knowledge acquisition techniques to facilitate the refinement of rule-based expert systems. At one level, learning can be viewed as a process of translation from an input representation into a target representation. However, generally it is desirable that the learning program do more than faithfully record all of the information presented to it. The purpose of a learning algorithm is to generalize, distill or abstract the input information into a more useful form. From the perspective of knowledge acquisition, machine learning programs such as ID3 (Quinlan, 1986), EITHER (Ourston & Mooney, 1990)), KBANN (Shavlik & Towell, 1989), FOIL (Quinlan, 1990), GOLEM (Muggleton & Feng, 1990), FOCL (Pazzani & Kibler, 1992; Pazzani, Brunk & Silverstein, 1991), FORTE(Richards & Mooney, 1991), and AUDREY (Wogulis, 1991) are just black box abstraction machines. Given a collection of examples (and possibly some background knowledge) each algorithm outputs a description of these examples in an alternate representation- a decision tree, a set of rules, or a network of weights. Usually the only control one has over these algorithm comes in the form of example selection and parameter specification. This limited form of control requires that the user have a detailed understanding of the learning algorithm. The interactive learning system, CLINT (DeRaedt & Bruynooghe, 1989;1992), is an example of an algorithm that begins to open the learning process. Learning in CLINT is guided by the user's answers to system generated queries. The MOBAL (Morik, 1991) environment extends the level of cooperation between the user and the learning algorithm even further offering a balanced cooperative approach in which the user is free to enter any information that the system might use. The interaction between the user and the learner is not rigidly dictated by the algorithm. Beyond MOBAL, FOCL-1-2-3 offers a balanced cooperative approach to knowledge acquisition and allows the user more control over the learning algorithm than previous systems.

We do not advocate eliminating the knowledge engineer from the knowledge acquisition process; in the near term we do not believe it is realistic for an unaided domain expert to interact directly with a knowledge acquisition tool to create a knowledge base. However, the creation of knowledge acquisition tools that provide a medium for communication between the domain expert and the knowledge engineer will help the knowledge base creation process. The balanced cooperative approach to learning is a large step in the right direction in terms of integrating machine learning and knowledge acquisition techniques. However, we feel that it is equally important to open the learning algorithm itself- making it more accessible and easier to understand. FOCL-1-2-3 is an extension of the multistrategy first order Horn-clause learning program FOCL that incorporates both of these approaches. FOCL-1-2-3 extends FOCL in three significant ways. First, we have

incorporated a graphic interface which allows dynamic visualization of the learning process. Second, we have integrated expert interaction into the low level learning algorithm by allowing the expert to select from a set of hypothesis. Third, we have extended the derivation structure that FOCL generates to include the examples that motivate each literal.

In addition to opening the learning process, we have added a number of features to FOCL that facilitate the knowledge acquisition process. The current generation of knowledge acquisition tools (e.g., Davis, 1978; Marcus, McDermott, & Wang, 1985; Wilkins, 1988) are useful in helping the user to discover why a single example is not classified correctly and in helping to identify a change to the knowledge base to correct this single example. However, they do not provide tools to assess the impact of this change on other examples that are correctly or incorrectly classified by the knowledge base. As a consequence, modifying a knowledge-base can be a time consuming activity in which changes in one rule produce unforeseen consequences that require changes in other rules. In contrast, the machine learning techniques we are employing use sets of examples to suggest revisions to the knowledge base. From the user's point of view, this technology simplifies the maintenance of the rule base of a knowledge-based system, by eliminating the iterative process of making a change based on a single example, and then discovering the impact of this change on other examples. In order to allow the user to manipulate and inspect a large numbers of examples in a coherent fashion, we have incorporated a spread sheet style interface.

The objective of this research is to make the modification of a domain theory analogous to the use of a spread sheet. That is, a change in the knowledge base should automatically update summary statistics describing which examples in a database are correctly classified by the knowledge base. Under user control, the specific examples that are affected by the change may be displayed. Furthermore, a change to the database should automatically update information on the accuracy and coverage of rules in the knowledge base. In addition (and unlike a spread sheet), changes to the database should result in the automatic creation of new rules (subject to expert inspection and confirmation) that abstract regularities from the data. Towards this end FOCL-1-2-3 maintains a close connection between examples and associated rules. Figure 1 illustrates how this information is currently made available to the user. The display indicated that eligible_for_deferment is true of eighteen people who are not required to pay back on their student loan and two people who are. The user has requested to see the examples which satisfy student_deferment, the rule responsible for covering the two negative examples. From this information, one can determine that three of the six students not required to pay back their loans are enrolled in 12 units, while the remaining students not required to pay back their loans are either enlisted in the armed forces (and thus eligible for a military deferment) or disabled (and thus eligible for a disability deferment). Furthermore, none

Figure 1. Finding the examples that satisfy the conditions of a rule.

of the students who are required to pay back their loans are enrolled in more than 11 units. This display gives a knowledge engineer familiar with the domain the information needed to revise the definition, by increasing the number of units required for a student deferment to more than 11.

In the remainder of this paper, we first describe the learning system, FOCL. Next, we discuss FOCL-1-2-3 and the extensions that have increased its utility as a knowledge acquisition tool. Finally, we discuss limitations of the system, outline the direction of our future work and draw preliminary conclusions about the approach we have taken.

2.0 FOCL

FOCL is a multistrategy relational learning program based on the inductive learning algorithm FOIL (Quinlan, 1990). In this section, we give a brief overview of FOCL for those unfamiliar with the system. For a detailed description of the FOCL algorithm and a complexity analysis see (Pazzani & Kibler, 1992).

FOCL uses a separate-and-conquer approach guided by an information-based heuristic to produce a set of clauses such that all the positive examples are covered by at least one clause and no negative example is covered by any clause. Even though FOCL employs a number of different learning strategies, it uses a single information-based evaluation function to guide learning within each component and to control the interaction between components. FOCL was designed to learn Horn-clause descriptions by combining analytical and empirical learning. As a consequence of this combination of learning methods, FOCL can utilize incomplete and incorrect domain theories (Mitchell, Keller, & Kedar-Cabelli, 1986; Rajamoney, & DeJong, 1987). FOCL uses the rule base of an expert system as the domain theory required by explanation-based learning and processes a collection of examples. FOCL extracts the portions of the domain theory that correctly classify examples, patches the portions of the domain theory that are useful but incorrect and discards the portions which have no information gain.

FOCL can be viewed as having two operators: **add-clause** which starts a new clause with the body true, and **add-literals** which adds literals to the end of the current clause body. FOCL performs the **add-literals** operator until no negative examples are covered by the clause. When a clause is formed, the positive examples covered by that clause are removed and FOCL performs the **add-clause** operator on the remaining examples, adding new clauses until all positive examples are covered by some clause. FOCL, like FOIL, uses an information-based metric to evaluate when to add a literal to a clause under construction. However, unlike FOIL, FOCL can use either an empirical (i.e., inductive) component or an analytical (i.e., explanation-based)

component to add literals to a clause. The result of this combination of methods is that the system can make use of domain knowledge when available to constrain the search for a concept definition. However, when the domain knowledge is incomplete or incorrect, the literals proposed by the analytical component will have less information gain than the literals proposed by the empirical component and new literals will be added by empirical methods.

2.1 Empirical Learning Components

The empirical learning components make use of three kinds of semantic knowledge to constrain search: argument type information (whether a specific argument must be of a specific sort), argument mode information (whether a specific argument must be bound or unbound) and argument constraint information (e.g., whether the predicate is commutative). In addition to checking all possible bindings of arguments to variables, it is possible for FOCL to consider argument bindings which contain constants.

2.1.1 Induction

Within FOCL, inductive methods for adding literals compute the information gain of every consistent binding of every relation and select the relation and argument binding with the highest gain. There are three variants of the induction component. The extensional induction component which is identical to FOIL performs induction over the relations defined as sets of positive and negative instances. The built-in induction component performs induction over relations defined as primitive lisp functions such as < and `equal`. The intensional induction component performs induction over the relations defined as rules. If an intensional relation has the maximum information gain, it is operationalized in the same manner that the target concept is operationalized by the analytical-component. This process is discussed in Section 2.2.

2.1.2 Cliché Instantiation

Clichés (Silverstein & Pazzani, 1991) are templates that specify relation and variablization restrictions on the generation of conjunctions of literals. Clichés differ from the rule models of MOBAL and the second-order schemata of CLINT in that they specify restrictions on both the relations which can be conjoined and on the arguments bindings of those relations. The cliché instantiation mechanism allows the empirical component a limited form of multiple step look ahead which aids FOCL's normal hill-climbing approach. The restrictions imposed by the cliché constrain the exponential explosion which would result from trying all pairs or triples of literals.

2.1.3 Determinate Addition

Muggleton & Feng (1990) introduced the notion of using determinacy and Quinlan (1991) incorporated a similar idea into FOIL. FOCL's implementation of determinacy is similar to FOIL's except that FOCL uses determinacy templates to create determinate literals. A determinacy template for a relation specifies a mode restriction and an associated determinacy. For instance the determinacy templates for the relation

father(?father ?child) = { (mike karen) (mike lynn) (jim lisa) (jim cynthia) (jim mike) (bob brit) }

would be (((:- :+) 1) ((:+ :-) 3)). This indicates that if ?child is specified then ?father can be determined uniquely, while if ?father is specified then in this data set there are at most three possible values for ?child. When all other learning components fail to produce a literal or conjunction with positive information gain, the determinate addition component adds all determinate literals of a specified determinacy to the current clause, then learning continues from this new state. The assumption is that the determinate literals will provide some bindings that further learning can take advantage of.

2.1.4 Simplification (Clause Generalization)

The simplification process can be invoked when FOCL has finished learning a clause, or when the analytical component has found a useful conjunction of literals. The simplification process deletes literals from the clause or conjunction that do not contribute to the definition. The deleted literals either unnecessarily exclude positive examples or exclude no negative examples. Simplification is helpful in generalizing overly specific domain knowledge, and in overcoming the effects of greedy search (Quinlan, 1990).

2.2 Analytical Learning Components

Like all explanation-based learning systems, FOCL's analytical component requires a target concept, (i.e., a non-operational definition of the concept to be acquired), a domain theory (i.e., a set of rules, such as the rule base of the expert system, that relates the non-operational definition to operational predicates), a set of operational predicates (i.e., features that the examples are expressed in terms of), and a set of classified training examples. FOCL computes the information gain of the target concept. If the target concept has positive information gain, then FOCL extracts the portion of the target concept that gives correct classification by an operationalization process. Otherwise, FOCL employs an empirical component. This strategy insures that FOCL uses as much of the user provided domain theory as possible, but disregards those parts of the domain theory that misclassify large numbers of examples.

In FOCL, the operationalization process differs from that of EBL in that it attempts to find a proof that is true of many positive examples and few negative examples rather than a proof of a single example. The operationalization process descends the path of maximum information gain from the target concept through the domain theory until it finds a conjunction of extensionally defined literals. These literals are then added to the clause under construction where they may be specialized by the empirical component (if the proof explains some negative examples.) This process is illustrated graphically in Figure 2.

3.0 FOCL-1-2-3

FOCL-1-2-3 is a knowledge acquisition environment based on FOCL that incorporates a graphic user interface that allows: dynamic visualization of the learning process, selection of alternate hypothesis to direct low level learning, and manipulation of both facts and rules. In addition FOCL-1-2-3 includes mechanisms to define examples in terms of relations and analyzing rule coverage. In keeping with the balanced cooperative approach it is possible to invoke the learning algorithm on any sub-problem in the knowledge base. The user specifies a learning task by selecting a set of examples and possibly a target concept. Alternatively, the user can enter definitions for new relations or modify existing relations at any time.

3.1 Visualizing the Learning Process

We believe that dynamic visualization of the learning process will prove helpful for anyone using FOCL-1-2-3. Instead of developing an abstract mental model of FOCL's learning process, there is an explicit graphic representation that the user can relate to. This explicit representation has made the learning algorithm much easier to observe and explain. We believe that integrating this kind of visualization technique into a knowledge acquisition tool can facilitate the exchange of information between the knowledge engineer and the domain expert.

Figure 2 is a snap shot of the operationalization process described in section 2.2. In the figure, there are 12 positive and 23 negative examples of no_payment_due remaining. There are two alternative ways that no_payment_due could be true (continuously_enrolled or eligible_for_deferment). The first is true of 9 positive examples and 13 negative examples, while the second is true of 8 positive and 9 negative. Because the second has higher information gain, FOCL recursively operationalizes it and finds that peace_corp_deferment and student_deferment both are true of 2 positive and no negative examples. The first is selected randomly for further operationalization and eventually the operationalization process returns the conjunction enlist(?0 ?1) and peace_corps(?1) which represents the rule that a person enlisted in the peace corps is not required to make student loan payments.

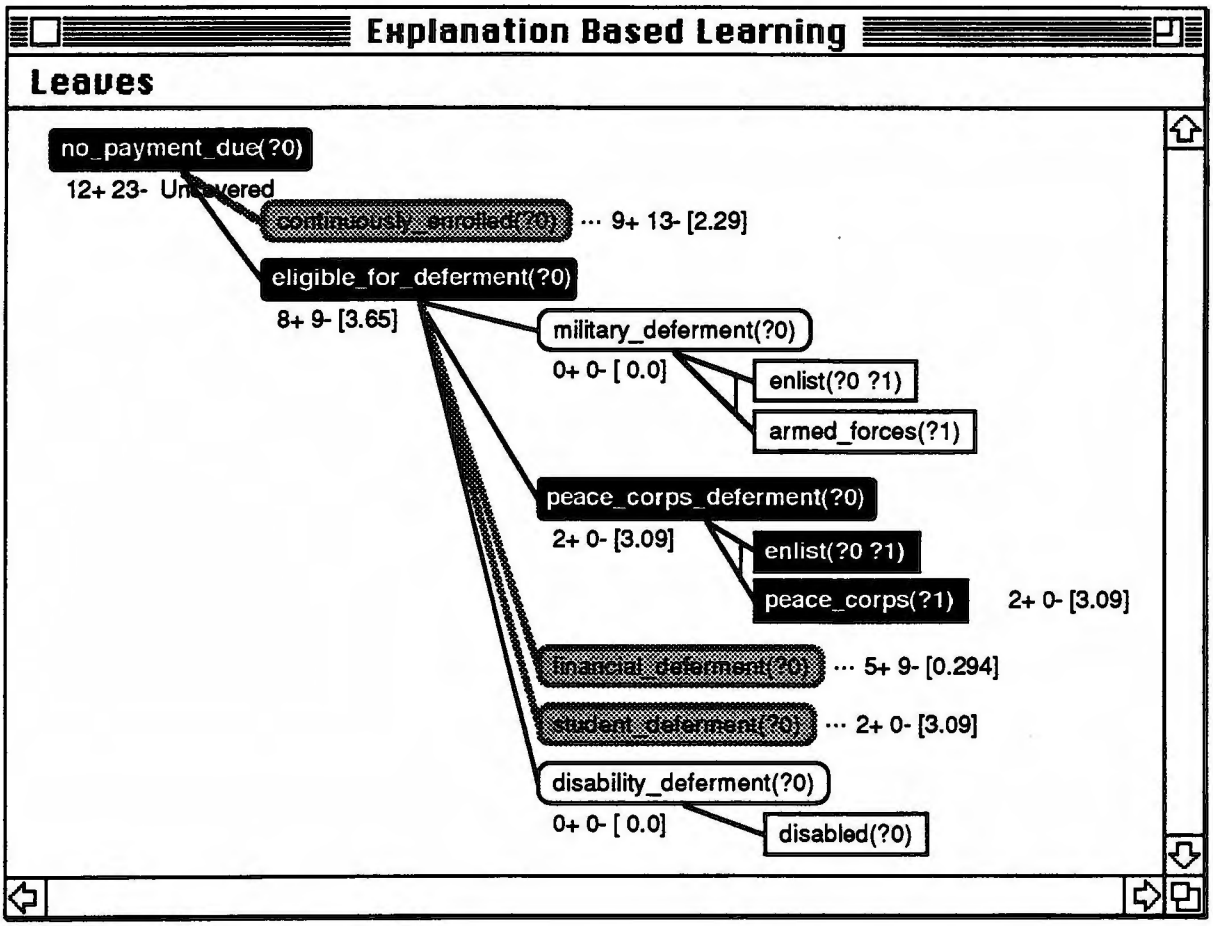


Figure 2. FOCL's Operationalizer

3.2 Examples and Relations

A subtle problem we had to address is that when dealing with a relational learning system the notion of an example is fuzzy. From the learning algorithm's perspective the universe is composed of relations. However, most people feel more comfortable dealing with examples. In attribute-value learning systems this distinction does not arise. Objects have a predetermined set of features. However in a relational learning system, objects as such do not exist, rather they are the values of a relation and are connected by other relations to other values. For instance, in an attribute value learning system (clifford-brown 12 UCI T) might be presented to the system as an example. While in a relational learning system the same information would almost certainly be distributed over a number of relations such as (name S14 clifford-brown) (enrolled S14 UCI 12) (male S14).

The primary feature that distinguishes relational learners from attribute value learners is their ability to access data which is not explicitly contained in an example. For instance, in the student loan

example, the fact than a school is a four year college is a condition that can be incorporated into a learned rule, while this fact is not about an individual student. A relational language provides additional expressiveness to the learned rules but also complicates the task of displaying an example, or even deciding what an example is. Should (four-year-university UCI) be presented to the user when information about student S14 is requested? Our solution is to supply FOCL-1-2-3 with example templates that represent user defined ways of viewing information from the database. Example templates tell the system what related information to present when displaying an instance of a specific relation. Example templates can be created manually or a default view is created automatically.

If the user asks to see some examples of a particular predicate the system retrieves the associated template, collects the related data and presents the user with the desired information. For instance in Figure 3, the system is displaying the examples of no_payment_due(?Person) that satisfy the literal enlist(?Person ?Organization). Although, the tuples that satisfy enlist(?Person ?Organization) are of the form (:person :organization) the example template for no_payment_due specifies that addition information including length of absence from school and employment status should be presented.

3.3 Manipulating Examples and Relations

In order to allow the user to deal with large numbers of examples in a coherent fashion, we have incorporated a spread sheet style interface into FOCL-1-2-3. The interface allows the user to inspect and manipulate a set of examples in a fashion which is analogous to a spread sheet.

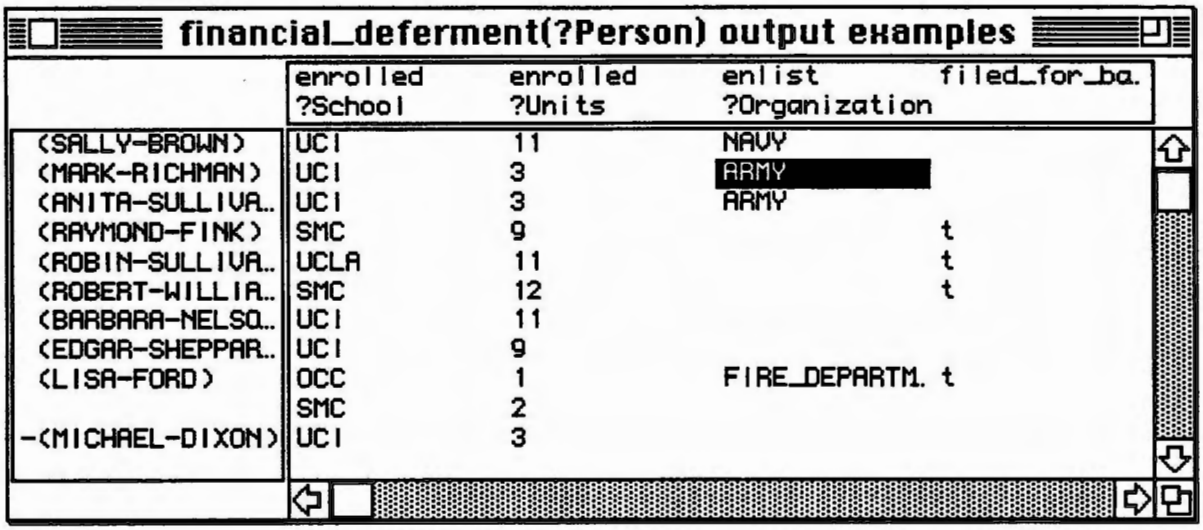


Figure 3. Spread Sheet Style Fact Interface

Figure 3 is a snapshot of this interface. Example labels are displayed in the box to the right of the window. Relation labels are displayed in the box at the top of the window. The first line of a relation label is the relation name and the second line is the relation argument. All of the information associated with an example appears in the same row as the example label. For instance the school Mark Richman is enrolled in can be found in row `MARK-RICHMAN` and column `enrolled ?School`. If there are multiple values for a given relation they appear as multiple entries. For instance, Lisa Ford is enrolled both at OCC and SMC. For relations arity two and higher relations the values of all none example label bindings are displayed in the table. Arity one relations are handled specially. Instead of displaying the argument bindings, the truth value of the relation is displayed. For example, filed_for_bankruptcy (lisa-ford) is a fact in the knowledge base and therefore the corresponding cell contains a `t`.

Using the spread sheet style interface, a value within a relation can be changed by selecting the associated cell in the display and entering the new value. All assertions associated with a given example can be retracted from the knowledge base by deleting the cell associated with the example. Similarly, relations can be deleted from the knowledge base by deleting the cell associated with the relation. In a future implementation, it will also be possible to add new examples and new relations to the knowledge base in a similar fashion. Once the modifications are complete, the changes are propagated throughout the system and the change in the coverage of any rule will be reflected.

In addition to the interface for manipulating examples and extensionally defined relations, we have created a graphic rule editor for FOCL-1-2-3. The rule editor displays a graphic representation of an intensionally defined relation. Using the rule editor a knowledge engineer can add literals to a clause, delete literals from a clause, reorder literals within a clause, add clauses to a rule, delete clauses from a rule, and change the rule's argument specifications. An innovative feature of the rule editor is that it retains the examples from the context in which it was invoked. Then the effect of any modification to the rule can be analyzed in terms of the coverage of these examples. For instance, Figure 4 shows summary information next to each clause. The specific examples can also be displayed (as in Figure 1). In cases where multiple uses of the same rule occur, the examples presented to each of these occurrence are collected and input to the editor. Since the editor environment is distinct from the knowledge base, the expert can evaluate alternate rule definitions without affecting the entire knowledge base. As with the example interface, once the modifications are complete the changes are propagated throughout the system.

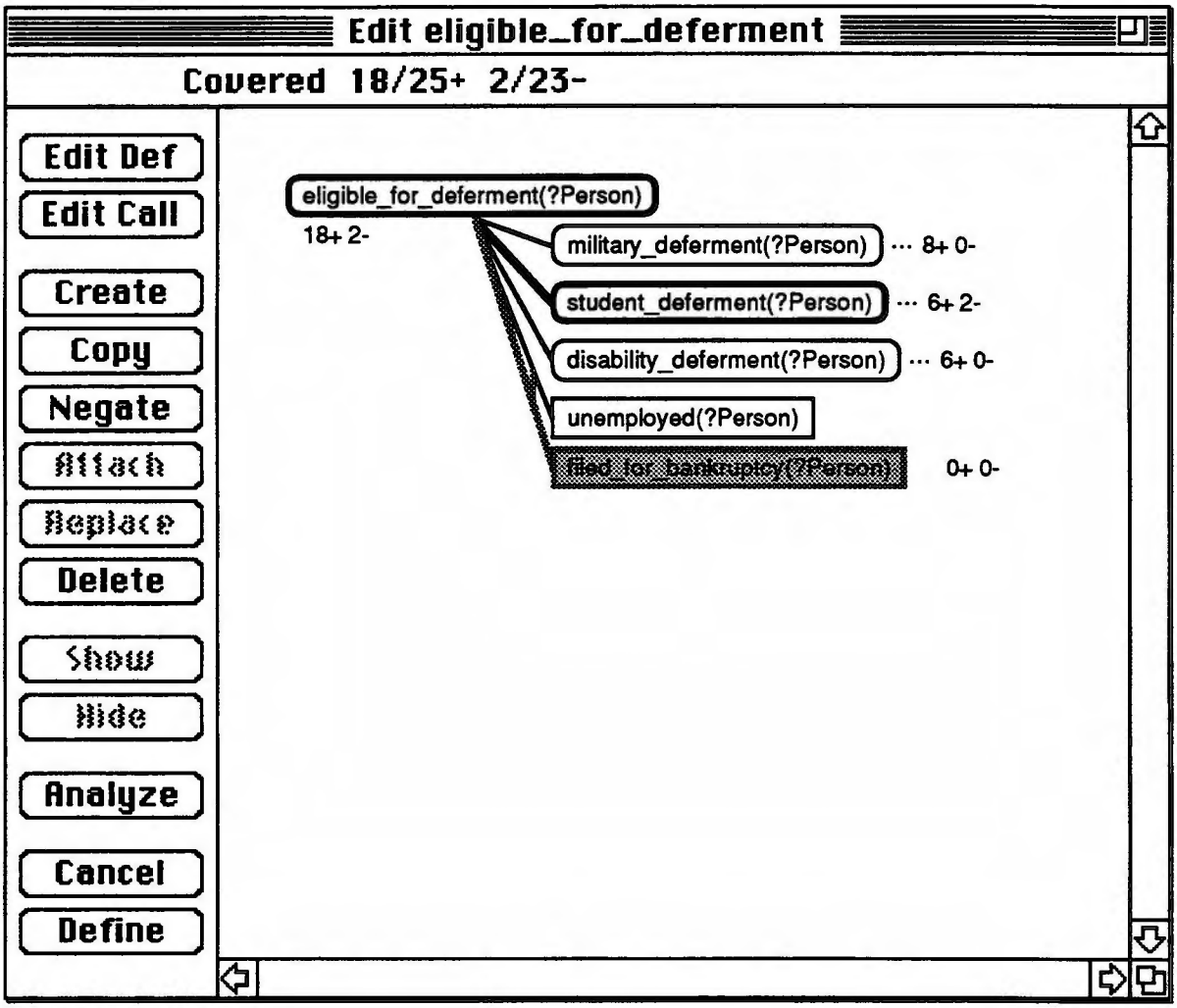


Figure 4. Graphic Rule Editor

3.4 Low Level User Interaction

We have extended the interaction between the user and the learning algorithm beyond the level of the balanced cooperative approach by allowing the user to interact with the learning algorithm at every level. Parameter setting, problem specification, and example selection is possible as it is with all learning algorithms. Algorithm directed interaction with the user (ala CLINT) is possible. Interactive modification of knowledge at anytime by user (ala MOBAL) is possible. Unlike other learning systems, FOCL-1-2-3 allows low level selection of the learning path, through a simple literal selection process. If so desired, expert interaction can be incorporated directly into the low level control structure of the learning algorithm. In this process, the learning algorithm explores the space of possible literals to add, collecting statistics on the examples covered by various

alternatives, but the selection of which literal to add to the clause under construction is performed by the knowledge engineer.

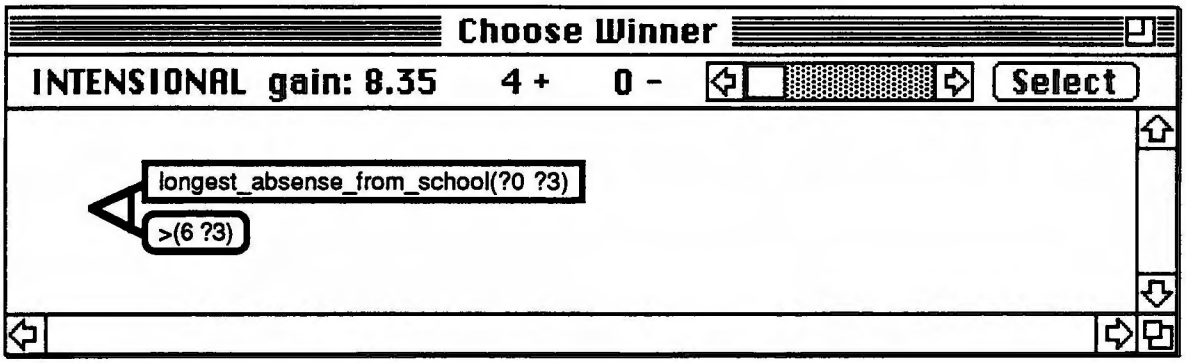


Figure 5. FOCL allowing expert to select from candidate extensions

The learning algorithm finds a specified number of the best candidates for extending the current clause from all the literals and conjunctions tested. The expert can then select the candidate to continue the learning process. The candidates are ordered according to decreasing information gain. In addition to the candidate, the candidate's gain, and the learning component that generated the candidate; the expert can see the clause being extended and the examples covered. We believe that this low level control, combined with the easy access to relevant information provides a useful extension to completely automated learning. Although information gain is very useful as a guide, there are times when an expert can provide insight, particularly when there is a semantic preference for some literal combination that cannot easily be captured in a selection function used by an automated learning system.

3.5 Analyze Coverage

To allow a knowledge engineer to explore the implications of changes to the knowledge base and to locate problematic portions of the knowledge base, we have created a tool that analyzes the coverage of the target concept. In Figure 6, the coverage analysis has been performed on the target concept no_payment_due using examples from the extensionally defined relation no_payment_due_fact. This tool displays the number of positive and negative examples that satisfy each clause and highlights literals that cover negative examples. After analyzing the coverage it is possible to display the set of examples that each literal covers. Reasoning abstractly about rules can be difficult, examples help to ground this process. In addition to maintaining the examples that each rule covers during analysis, FOCL also maintains the coverage generated at learning time and allows the expert to access this information.

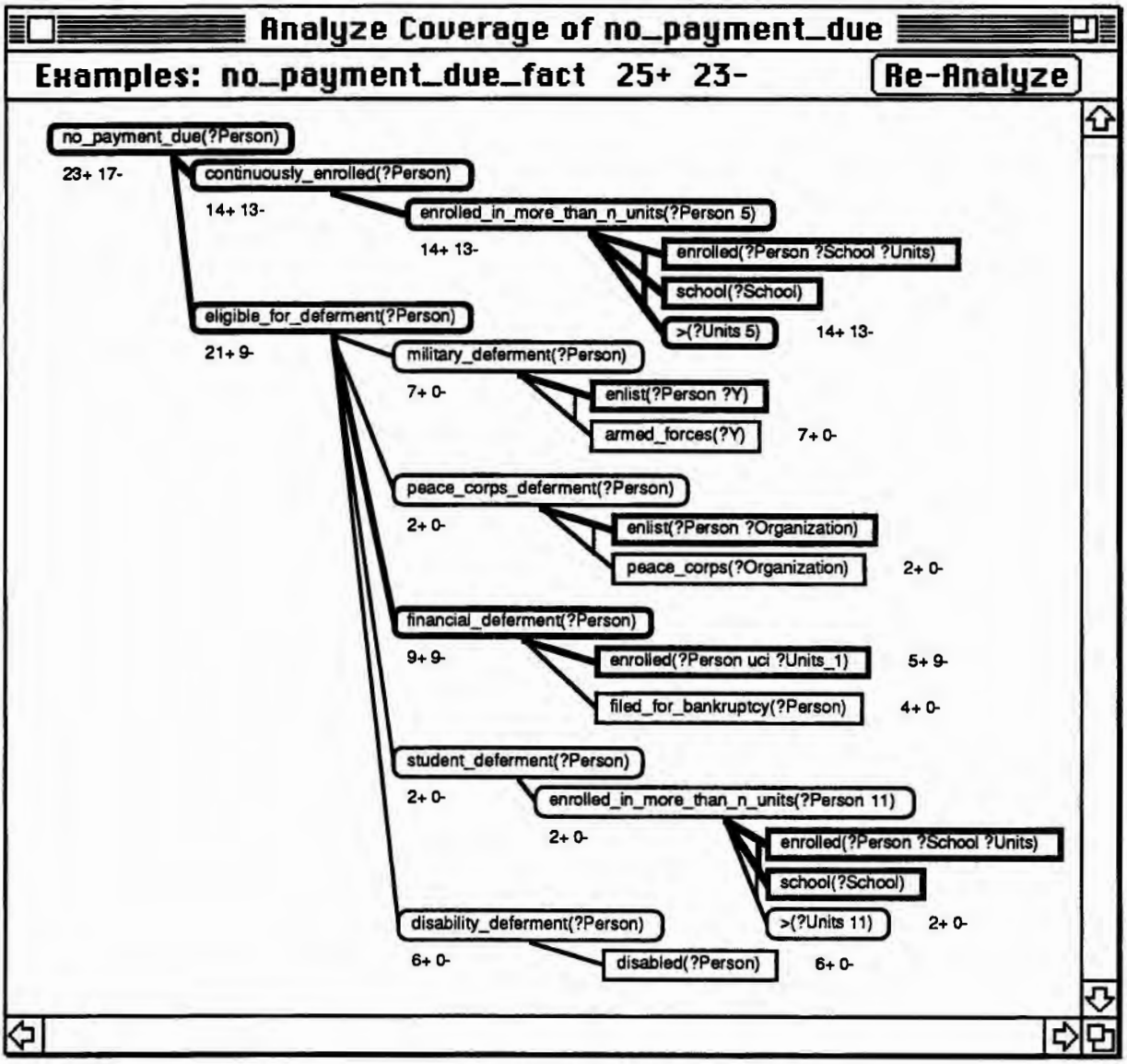


Figure 6. Analysis of Goal Concept Coverage

3.5 Knowledge Revision

Previously we reported on a prototype knowledge acquisition tool KR-FOCL (Pazzani & Brunk, 1990) that partially automates the task of identifying rules responsible for errors in expert systems. KR, the knowledge revision component, was designed to use three heuristics and a derivation of the learned descriptions generated by FOCL to suggest revisions to the knowledge base. KR has been integrated into FOCL-1-2-3. The interface to KR is a specialized version of the rule editor. After learning KR can be invoked.

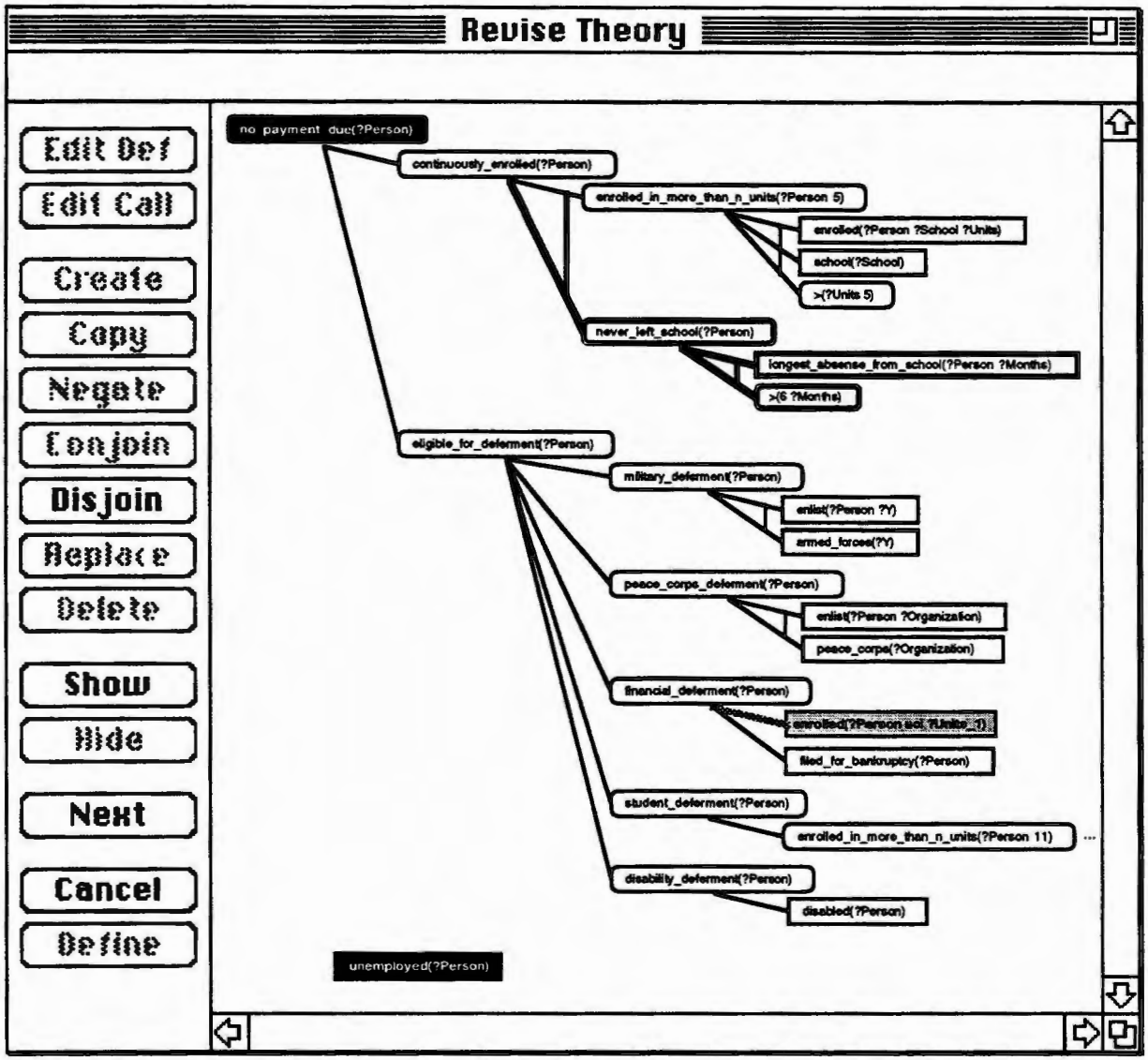


Figure 7. Knowledge Revision using KR

First, KR collects all the clauses from the learned description that were learned via a combination of empirical and analytical techniques. It then clusters those clauses that were learned by operationalizing the same path through the domain theory extracting the empirical literals from the clauses in each cluster. Sequentially, the operationalized path and empirical literals from each cluster are presented to the user as disjoint graphs. Empirical literals from the same clause appear as conjunctions, while literals from different clauses appear in separate disjuncts. The default attachment point from the learned description is indicated. The user is able to accept the indicated attachment, select an alternate conjunction to which the empirical literals will be conjoined, or move on to the next revision suggestion. In Figure 7, the user has conjoined the induced clause never_left_school(?Person) to the operationalized literal enrolled_in_more_than_n_units(?Person).

Next, KR collects all the clauses from the learned description that were learned via empirical methods alone. Again the user is presented with two disjoint graphs only this time the user is asked to select an attachment point for which the empirical literals will form a new disjunct. Figure 7 illustrates this. During learning the induced clause unemployed(?0) was learned as a disjunct for no_payment_due(?0). The knowledge engineer is presented with this default attachment point and given the option selecting another.

Finally, KR suggests deleting literals and clauses that were not needed to obtain correct classification. These include literals that were removed during a simplification process and portions of the domain theory that were never operationalized.

FOCL-1-2-3 addresses some of the limitations of KR-FOCL. KR-FOCL did not explain why it recommended certain revisions. FOCL-1-2-3 maintains a more complete derivation which includes the examples that satisfy each literal. Combined with a description of the revision heuristics these examples provide an explanation of the revision recommendation. Instead of serially asking verification questions about a large number of possibilities as KR-FOCL did. FOCL-1-2-3 presents the expert with a graphic editor and a suggested revision. The expert can choose to accept the revision as is, propose a different revision, or skip to the next suggestion. Additionally, we have begun to add capabilities similar to programs such as SEEK (Politicks & Weiss, 1984) that can quickly assess the impact of revision. It is no longer necessary to re-run FOCL or re-analyze the entire target concept to determine the impact of a change. Re-analysis is performed automatically and limited to the affected portions of the knowledge base.

3.6 Other Tools

Most of FOCL-1-2-3 is designed to help the user identifying logical errors in a rule base. However, it also checks the syntactic structure of the rule base. Currently, four types of errors are detected: using any predicates that are not defined, using the wrong number of arguments in a predicate, using arguments that are not consistent with the type declaration and making sure each variable in a clause is used more than once. If any of these errors are detected, FOCL-1-2-3 collects all warnings about a rule, and opens up an editor window that allows the knowledge engineer to see the warning messages and to correct the rule definition. Figure 8 illustrates one such example.

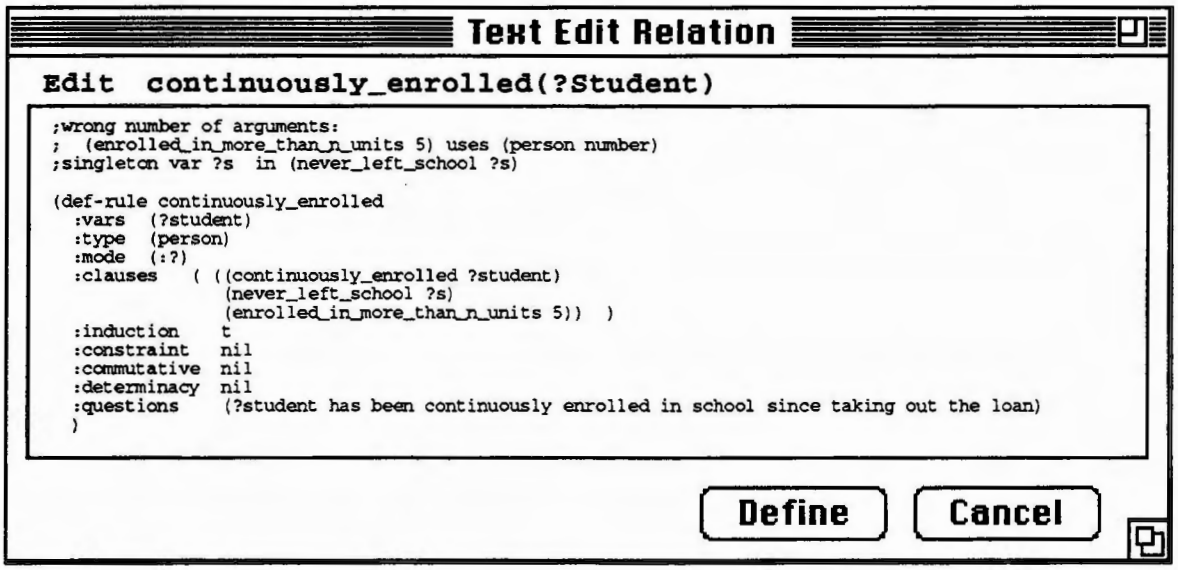


Figure 8. Editing a rule with stylistic warning

4.0 Limitations and Future Work

FOCL-1-2-3 is limited by the underlying machine learning technology; it is designed to work only with rule-based expert systems that perform classification tasks. If the problem cannot easily be expressed using Horn-clauses or if the rules require certainty factors or if the expert system relies on a complex conflict resolution strategy to determine the applicability of rules, then FOCL-1-2-3 is not an appropriate tool to use. If FOCL cannot solve the learning problem by using hill-climbing search, then FOCL-1-2-3 reduces to a fancy editing environment. We are currently investigating extensions to the analytical component that will allow FOCL to learn from a larger class of erroneous domain theories. One approach we are investigating is the use of a mechanism which allows disjunctive frontiers to be operationalized in a single pass rather than requiring the operationalization of conjunctive paths in isolation. We are also investigating the integrating abductive techniques to localize blame and direct learning of intermediate concepts.

As a theory revision tool FOCL-1-2-3 is limited in that it does not perform revisions autonomously, instead it presents revision suggestions to the user who performs the actual modification. However, the new operationalization strategy mentioned above offers a natural mechanism for performing revision at learning time. We plan to investigate the utility of this approach compared to the current approach of revision only after learning a complete concept definition.

Another of FOCL's limitations is that learning occurs at a very syntactic level. Currently, the only semantic knowledge that FOCL utilizes comes in the form of a limited ontology of relations used by the cliché instantiation mechanism and a rudimentary type system. By extending the ontology and integrating it into the other learning components, it would be possible using a relevance metric to decrease the space that FOCL searches. Similarly, by enhancing the sort system to handle hierarchical typing, the search space can be reduced thereby increasing the effectiveness of the learning components.

FOCL-1-2-3 does not address the problem of knowledge integration. Since the system does not handle structured objects, knowledge integration usually devolves to relation modification. However, sophisticated knowledge integration techniques like those found in KI (Bareiss, Porter & Murray, 1989) can be incorporated as we begin to deal with more complex representation languages and structured objects.

Currently, we have only tested FOCL-1-2-3 on a small number of knowledge bases: the student loan domain, a trade negotiations domain, and a track identification domain. FOCL-1-2-3's interface is Macintosh dependent, as a result the size of the domains are limited. However, the underlying learning algorithm which is machine independent has been tested on larger problems and appears quite robust. We are in the process of expanding FOCL-1-2-3's repertoire.

We are developing a protocol to evaluate the effectiveness of the extensions in FOCL-1-2-3. We plan to conduct two experiments in the context of an upper division undergraduate class on expert systems. The first experiment will assess the new learning-revision mechanism's ability to perform autonomous theory revision and compare it to the students' ability. The second experiment will evaluate the utility of the learning-revision mechanism in the knowledge acquisition process. Students will construct a knowledge base using FOCL-1-2-3 with and without the learning-revision component. The creation time, accuracy and generalizability of each knowledge base will be compared. Systematically evaluating the utility of each of the knowledge acquisition extensions incorporated into FOCL-1-2-3 is important, but the time required to do so is prohibitive. Thus, we are focusing on what we feel to be the most important component. However, we intend to survey the class to determine their feelings about the utility of various components.

5.0 Conclusions

In a large system, it is difficult for a user to anticipate the impact that a change made to the knowledge base will have on the coverage and correctness of the classification of examples in a database. We have attempted to address this issue by developing graphical methods to summarize the effect of changes. FOCL-1-2-3 is a further step in the integration of knowledge acquisition and machine learning techniques. It offers an open learning algorithm and a balanced cooperative approach to knowledge acquisition while incorporating a number of novel features including:

- a dynamic graphic representation of the learning process,
- a rule editor that can assess changes to a rule in terms of local example coverage,
- a spread sheet like interface for manipulating examples and facts,
- a mechanism for quickly assessing the affect of modifications to the knowledge base, and
- a technique for maintaining a connection between rules and the examples they cover.

This latter technique is useful for providing a context in which the expert can reason about rules. In the future these examples will provide the basis for automation of the revision process. We envision a revision component that interacts with the expert in a fashion similar to the learning component. The revision component will analyze a myriad of possible revisions and cull from among them a specified number of the best that will be suggested to the knowledge engineer.

One feature that distinguishes FOCL from many other learning systems is its ability to perform recurrent learning. FOCL can utilize the descriptions it learns as background knowledge and is therefore capable of assisting in complex theory formation tasks. As a trivial example, assume that FOCL had not been provided with an intensional definition of continuously_enrolled, but had been provided with a set of examples of students who were continuously_enrolled. FOCL could be used to learn an intensional definition of continuously_enrolled. This rule could then be used as background knowledge when learning a more complex concept like no_payment_due or eligible_for_scholarship. Although, this type of theory formation process is difficult to control automatically, it is simple to direct manually, and it is a natural nexus between knowledge acquisition and machine learning. We believe that FOCL-1-2-3 provides an excellent environment for exactly this type of activity.

References

Bareiss, R., Porter, B., & Murray, K. (1989). Supporting Start-to-Finish Development of Knowledge Bases. *Machine Learning, 4*, 259–283.

Davis, R. T. (1978). Model-directed learning of production rules. In D. Waterman & F. Hayes-Roth (Eds.), *Pattern-directed inference systems*. New York: Academic Press.

De Raedt, L., & Bruynooghe, M. (1989). Towards friendly concept-learners. *Proceedings of the Elenenth International Joint Conference on Artificial Intellegence*. Morgan Kaufmann.

De Raedt, L., & Bruynooghe, M. (1992). Interactive Concept-Learning and Constructive Induction by Analogy. *Machine Learning, 8*, 107–150.

Marcus, S. McDermott, J. & Wang, T. (1985). Knowledge acquisition for constructive systems. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*. Los Angeles, CA: Morgan Kaufmann.

Mitchell, T., Keller, R., & Kedar-Cabelli, S. (1986). Explanation-based learning: A unifying view. *Machine Learning, 1*, 47–80.

Muggleton, S. & Feng, C. (1990) Efficient induction of logic programs. *Proceedings of the First Conference on Algorithmic Learning Theory*, Tokyo, Japan. Ohmsha.

Morik, K. (1991). Balanced Cooperateive Modeling. *Proceedings of the FirstInternationWorkshop on Multistrategy Learning*. 65–80. Harpers Ferry.

Ourston, D., & Mooney, R. (1990). Changing the rules: A comprehensive approach to theory refinement. *Proceedings of the Eighth National Conference on Artificial Intelligence* . 815–820. Boston, MA: Morgan Kaufmann.

Pazzani, M., & Brunk, C. (1990). Detecting and correcting errors in rule-based expert systems: an integration of empirical and explanation-based learning. *Proceedings of the Fifth Knowlegde Acquisition for Knowledge-Based Systems Workshop*, Banff, Alberta: SRDG Publications.

Pazzani, M., Brunk, C., & Silverstein, G. (1991). A knowledge-intensive approach to learning relational concepts. *Proceedings of the Eight International Workshop on Machine Leanring*. 432–436. Evanston, Il: Morgan Kaufmann.

Pazzani, M., & Kibler, D. (1992). *The utility of knowledge in inductive learning* . *Machine Learning, 9*, 57–94.

Politakis, P & Weiss, S. (1984). Using empirical analysis to refine expert system knowledge Bases. *Artificial Intelligence, 22*, 23-48.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*, 81-106

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning, 5*, 239–266.

Quinlan, J. R. (1991). Determinate Literals in Inductive Logic Programming. *Proceedings of the Eight International Workshop on Machine Leanring*. 442–446.Evanston, Il: Morgan Kaufmann.

Rajamoney, S., & DeJong, G. (1987). The classification, detection and handling of imperfect theory problems. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* . 205-210. Milan, Italy: Morgan Kaufmann.

Richards, B. & Mooney, R. (1991). First-Order Theory Revision. *Proceedings of the Eight International Workshop on Machine Leanring*. 447–451. Evanston, Il: Morgan Kaufmann.

Shavlik, J., & Towell, G. (1989). Combining explanation-based learning and artificial neural networks. *Proceedings of the Sixth International Workshop on Machine Learning*. 90–93. Ithaca, NY: Morgan Kaufmann.

Silverstein, G., & Pazzani, M. (1991). Relation clichés: Constraining constructive induction during relational learning. *Proceedings of the Eight International Workshop on Machine Learning*. 203–207. Evanston, Il: Morgan Kaufmann.

Wogulis, J. (1991). Revising Relation Domain Theories. *Proceedings of the Eight International Workshop on Machine Learning*. 462–466. Evanston, Il: Morgan Kaufmann.

An Operational KADS Modelling Language and Tool Support for its Application

SIMONE BÜRSNER[1]
University of Heidelberg
Institute for Medical Biometry and Informatics
Department of Medical Informatics
Im Neuenheimer Feld 400
W-6900 Heidelberg, FRG

THOMAS WETTER
IBM Germany
Science Center, Institute for Knowledge Based Systems
Wilckensstr. 1a
W-6900 Heidelberg, FRG
EARN: WETTER at DHDIBM1

Abstract

KADS[2] as a method for structured development of knowledge based systems regards knowledge engineering as a modelling process. In this paper the focus is directed on knowledge acquisition. KADS is a major representative of a group of approaches where knowledge acquisition is looked upon as a modelling and not as a mining activity. In order to support modelling during knowledge acquisition FORKADS[3] - an executable formal language for KADS conceptual and interpretation models - is presented here. The language FORKADS is based on order sorted predicate logic. It can be applied for writing KADS conceptual and interpretation models. A conceptual model written in that language should be understood as a highly accurate specification but not as final code. Both for facilitating the application of FORKADS and for handling real world knowledge bases, tool support is mandatory. The FORKADS-tool presented in this paper consists of two components: a graphical user interface and a generator producing operational models, coded in Prolog.

1 Introduction

1.1 Principles of KADS

After seven years of ESPRIT funded research, **KADS** - **K**nowledge **A**cquisition **D**ocumentation **S**tructuring [WIELINGA, BREDEWEG, and BREUKER, 1987] - has evolved into the most influential approach to knowledge engineering in Europe (cf. [WIELINGA, SCHREIBER, and

---

1 The research reported in this text was conducted in: IBM Germany, Science Center, Institute for Knowledge Based Systems, Wilckensstr. 1a, W-6900 Heidelberg, FRG.

2 **K**nowledge **A**cquisition and **D**ocumentation **S**tructuring; ESPRIT project P1098; contractor: University of Amsterdam, J. BREUKER and B. WIELINGA

3 FORKADS is intended to remind of both: **for**mal **KADS** and **for** (= in support of) **KADS**.

BREUKER, 1992]). Its influence can also be found in other systematic approaches to knowledge engineering such as [O'NEIL, GLOWINSKI, and FOX, 1989] and [LINSTER, 1989]. KADS tries to combine principles from software engineering and new achievements from knowledge engineering. Some of KADS' distinguishing features are that it aims at being domain independent (in contrast to domain related workbenches such as OPAL [MUSEN et al., 1987] or KNACK [KLINKER et al., 1988]) and independent of knowledge representation formalisms (in contrast to e.g. PRED [XIE, DUMARESQ, and WINNE, 1988]) and it supports re-use of parts of a solution. The latter aspect has been intensely treated in a report on one of the commercially successful application projects of KADS (cf. [PORTER, 1992]).

As a method for ***structured*** development of knowledge based systems KADS is life cycle oriented and intended to support all development activities methodologically[1]. KADS regards knowledge engineering as a ***modelling*** process and clearly distinguishes several activities, each producing a special kind of model.

Concretely KADS speaks of
- a **conceptual model** as a knowledge level model of the domain expertise,
- an **abstract design model** as a functional description of the artefact to be built, and
- a **technical design model** as a detailed description in terms of design elements of the artefact to be built.

We concentrate on the first activity - knowledge acquisition and precise specification of the knowledge that characterizes a field of expertise - well knowing that a complete methodology must cover the entire process from requirements analysis to implementation and maintenance. After several stages of informal collection of manifestations of expert knowledge during knowledge elicitation, analysis, and interpretation, the knowledge engineer has to end up with a ***precise specification*** of expertise: the **conceptual model**.

Other principles of KADS are described in [WIELINGA, SCHREIBER, and BREUKER, 1992].. They do not play a role for the subject treated in this text.

1.2 Constituents of the Conceptual Model

In KADS the conceptual model is organized in four layers:

domain	application specific static knowledge
inference	elementary application independent methods
task	major subproblems to be solved in the domain
strategy	control of solution process

Identifying concise sets of generic elements for these layers has always been among the research goals. Starting from a KL-ONE related epistemology (cf. [BRACHMANN and SCHMOLZE, 1985]) for the **domain layer** [SCHREIBER et al., 1988] identify *concepts*, *relations*, and *complex structures*. On the **inference layer** we have *knowledge sources* and *metaclasses*. Knowledge sources realize elementary inferences which apply to roles in the

[1] For example design activities are not supported by the KADS methodology at the moment.

inference process (hypothesis, set of alternatives, solution etc.), named metaclasses. The metaclasses are dynamically filled by concepts from the domain. Knowledge sources can be seen as functions that transform, select, create etc. entities. On the **task layer** the language requires *goal statements* and *control statements*. Elementary goals may correspond (conceptually) to application of functions, i.e. immediate use of knowledge sources. Non-elementary goals correspond to respective subtasks. This is superimposed by more or less conventional control constructs such as loop, bifurcation etc. embedding the goal statements. In KADS-I the strategy layer has never really been used, whereas the KADS-II perspective requires some research in reflective systems but is not yet mature for application. That is why the strategy layer is not treated here.

1.3 Purpose of the Conceptual Model

From the perspective of knowledge acquisition as the first major phase, the conceptual model is the final document which is produced on the basis of results from elicitation sessions and which fixes every identified item that was judged to be relevant for covering the contents of the application or the expertise to be modelled. In other words, conceptual modelling is interwoven with elicitation. In the present text we regard a conceptual model as a result that emerges from elicitation ***and*** formalization, but do not treat elicitation.

From the perspective of system design, i.e. the second major phase, the conceptual model is the mandatory input concerning the involved knowledge. In other words, the conceptual model is ***the*** interface between content related acquisition activities and artefact related design activities. This, however, means that only the entire contents and nothing more must be fixed ***unambigously*** in the conceptual model[1]. It should not be taken as the operational system but as an executable specification, from which a design still has to be derived.

The main reasons why FORKADS conceptual models, although operational, should not be considered as final systems are that

- the specification names all elements and provides definitions of the functionalities, but it does by no means optimize data structures, access methods, algorithms etc.
- it does not make explicit the individual integration need with other software components
- the FORKADS interpreter[2] is general purpose, i.e. operational for a wide variety of theories and queries, hence cannot even be close to optimal for specific needs of one application

1.4 Interpretation Models

KADS provides conceptual models *"with an empty domain layer"* to supply the knowledge engineer with *"a template for a new domain"* ([WIELINGA, SCHREIBER, and BREUKER, 1992],

---

1 This may include deliberate decisions before or during knowledge acquisition to leave out certain details that are either judged irrelevant or resist modelling.

2 which is based on the LILOG interpreter, cf. section **3.1 Realization**, ***LCONMOD*** in this paper and [BOLLINGER and PLETAT, 1991]

p. 29). Interpretation models are task structures which a new application can be matched to in order to ***guide*** the process of ***understanding*** its characteristics. Interpretation models are formulated in conceptual (as opposed to data processing) terminology, i.e. they aim at being ***descriptive*** of the phenomena that are being modelled. In case of a perfect match between a given application and an existing interpretation model, the interpretation model gets a flavour of a re-useable element, although technical re-usability of building blocks is not a major theme in KADS.

The constituents of interpretation models, introduced above as knowledge sources or elementary inferences, are also aimed at being as general as to be used in many interpretation models. All these repetitive elements contribute to the learnability of KADS. Its models for the first activities use application (as opposed to data processing) terminology. The constituents of the models are intended to form a generic base set for which re-use is emergent.

Interpretation models exist for systematic classification, heuristic diagnosis, monitoring, assessment etc.. We preserve this notion and its name and give the aspect of 'what inferences match which control constructs' a more precise meaning. A decision about a control structure involving task and inference layer goes with a decision about a domain structure. It is e.g. useless to formulate a loop over subconcepts of a concept when a concept is elementary, i.e. does not have any subconcepts.

This points into the direction of an expansion of the notion of interpretation model towards the notion of **stereotype** which (among others) comprises a formalized interpretation model to describe control flow and a 1-1 related sort model to describe how concepts in a domain have to be structured for an interpretation model to be applicable.

1.5 Requirements to Support KADS Conceptual Modelling

- For a precise specification which fulfills the central function of the conceptual model (described in 1.3) we need ***formal languages***.
- For making sense and offering a modelling help we have to supply ***meaningful templates*** (KADS: *interpretation models*).
- For handling real world knowledge bases ***software tools*** are mandatory.

1.6 Scope of this Paper

We present FORKADS - an executable formal language for KADS conceptual models - and the lessons learnt from developing the language. Furthermore we describe the components of a software tool which facilitates building conceptual models in the formal language FORKADS. The formalization of interpretation models has already been presented in [WETTER and SCHMIDT, 1991] based on a former version of FORKADS. More foundational details about knowledge representation aspects and an example illustrating the application to a task in the field of banking are in preparation (cf. [WETTER, 1992]).

2 FORKADS - An Executable Formal Language for KADS Conceptual and Interpretation Models

We have argued that, due to their role in subsequent phases of the knowledge engineering life cycle, conceptual models should be formal. This may appear contradictionary to the notion of conceptual models as human conceptualizations of technical devices. If the emphasis is on 'human conceptualization', the contradiction exists. If the emphasis is on 'technical devices' such as software systems, and if from the conceptual model the behaviour of the software system should be predictable, formal notation is indeed required. This is the case in knowledge engineering à la KADS. Still there may be a number of different formal languages. FORKADS is one of them, as opposed to collections in KADS documents (cf. [WIELINGA, SCHREIBER, and BREUKER, 1992])[1].

2.1 Basic Arguments About Design of the Formal Language

All informal notations require interpretation instead of transformation into design elements and hence are error-prone due to mis-interpretations. If it is mandatory to avoid mis-interpretations, a formal language lends itself to fix the contents unambigously.

When written in a formal language, the meaning of all expressions of a conceptual model is determined by a formal semantics, shared by all individuals writing and using conceptual models.

Formalization will inevitably have to abstract from one or the other detail encountered during knowledge elicitation, depending on the expressiveness and structure of the formal language. That is why care must be taken not to lose too much of the modelling power of KADS.
Though reduction is an inevitable part of formalization, the question still is ***when*** to reduce and ***how much*** to reduce. For enabling job and skill sharing - which means knowledge acquisition and system design are done by different persons - it is mandatory to reduce during conceptual modelling and not during design. Otherwise the system designer would, maybe unconsciously, make decisions about the contents of an application for which he is neither skilled nor informed nor can be made responsible. In order to preserve similarity to the KADS methodology and not to hinder conceptual modelling, we must use a formal language of highest possible expressiveness (criterion to follow). More specifically, we demand structural accordance between language and method, that means a formal language must supply expressions to cover KADS layers and modelling principles.
In other words: The structure of the respective formal language is decisive for its applicability. It will be argued that the structure of the presented language closely follows KADS recommendations about how to arrange models of expertise. Because these recommendations have already proved useful in application projects there is good hope, though not a stringent argument, that the suggested language also enables competent human use.

---

[1] Cf. variety of uses of 'select' as a striking example of underspecification. This underspecification which implies that the meaning of KADS inferences is not really known, makes it impossible to compare them one-to-one to FORKADS inferences.

Nevertheless a FORKADS model is a reduced or abstracted notation of some expertise. Deviations can be documented by elicited items which lead to deviating expressions in the model. This process is, however, not supported in the present FORKADS, because documents from elicitation sessions are not dealt with.

2.2 Towards Concrete Structural Decisions for FORKADS

Operational semantics allows to build interpreters to simulate the behaviour of models, which delivers further insight into their properties and enables evaluation of models. This is very important for knowledge engineer, expert, and designer.

However there exists a trade-off between
- expressiveness (as required in section 2.1)
- and operational semantics (as required in this section).

Such a trade-off can best be dealt with where a theory exists which combines denotational and operational semantics. For fields of expertise with predominant occurrence of processes and their mutual control extensions of Petri nets such as Pr/T-nets[1] [cf. BRAUER, REISIG, and ROZENBERG, 1987] might suggest themselves. When we take into account usability within the methodology KADS as a further criterion, it has turned out (cf. [WETTER, 1992]) that among the highly expressive yet executable languages order sorted logic, superimposed by procedural control mechanisms, well suits the prescriptions made through the three layers of KADS conceptual models.

Details about this language will follow now; a software tool to build and work with models in that language will be presented in the next chapter.

2.3 Constituents of the FORKADS Layers

We go through the catalogue of language elements in the sequence from domain via inference to task layer. Short reference to the required constructs for the KADS conceptual model has already been given in section 1.2. We aim at a ***closed*** formalization of the KADS layers. This requires very complex language constructs and dedicated means for connecting the layers.

For illustration of the essential language elements of the three layers we use examples from the ***stereotype assessment.*** We applied this stereotype for building a conceptual model of a concrete banking application - balance analysis - where customers are judged whether they are still credit-worthy or not.

---

[1] Predicate/Transition nets

2.3.1 The Domain Layer

For the representation of application specific knowledge we distinguish four types of static information: **concepts, instances of concepts, relations, and complex structures.** Subsequently we introduce the systematic means and the ways how they can be combined.

Concepts and their instances

Facing the need to represent concepts and their instances in a logic environment, it suggests itself to represent instances as constants which can be stored in variables and concepts as sorts. This allows to argue about sort-subsort relation and alike in a natural way. Further advantages will become evident when introducing the basic classes of inferences. Hence an ***instance of a concept*** can be stored in a ***logic variable*** (starting with an uppercase letter) which must be of some ***sort*** (starting with a lower case letter) in the programming-in-logic environment. Generally, we require

```
<Variable_name> : <sort_name>
```

In the sort model for the abstract concepts of the stereotype assessment, we define

```
Entity : system_model
```

Concrete instances that may 'fill' variables are introduced into the language as ***reference objects*** or ***refos*** by simple expressions:

```
refo <constant_name> : <sort_name>
```

Relations

A ***structure of a domain*** must be specified in terms of its general expressions, which are the sorts. Inasmuch as structure of a domain means relations between its concepts, a number of relationships between sorts is available from [PLETAT, 1990]. First there is the ***subsumption relation*** which entails certain inheritence properties:

```
sort <sort_name_1> < <sort_name_2>
```

For modelling a concrete domain further structuring elements prove very useful. Therefore we introduce two types of properties of a sort from the KL-ONE tradition, which do not directly occur in [SCHREIBER et al., 1988]. However [SCHREIBER et al., 1988] and other KADS publications also quote KL-ONE as one of their influencing roots. Concretely, we have one-place functions named ***features*** and two-place relations named ***roles*** (corresponding to KL-ONE RoleSets), attached to a sort:

```
sort <sort_name> < supersort;
   features
      <feature_name> : <target_sort_name>,
         ...
   roles
      <role_name> :: <related_sort_name>,
         ...
```

In the stereotype assessment, these substructures among others occur as

```
sort system_model < top;[1]
   features
      identifier_ : top,
         ...
   roles
      indicator_ :: indicator,
      observable_ :: observable,
         ...
```

In order to be syntactically correct, indicator and observable must themselves be defined as sorts.
Concepts, their properties and to what other concepts they have what kind of relatedness, can to a large extent be expressed by the above means of sorts, features, and roles. To preserve a certain flexibility to write more complex relations without much of an increase in complexity of the required expressions, we may also use ***n-place predicates***. The general form to introduce a predicate is:

```
predicate <predicate_name>( <argument_name_1> : <sort_1>, ...,
                            <argument_name_n> : <sort_n> ).
```

The expressions argument_name_i have no structural meaning. Their only purpose is to allow reference to arguments explicitly by name (instead of implicitly by place). sort_1 through sort_n, however, imply that the respective concepts are in a relation named predicate_name. An axiom (to be introduced below) comprises among others the concepts (represented by sorts) system_model, indicator, and numeric in a relation with the predicate name indicator_formula:

```
predicate indicator_formula( candidate : system_model, ind : indicator, eval : numeric ).
```

Complex Structures

Now that we have given a first impression how to write properties about the concepts used in the sort model of the stereotype assessment, we need some more detail on how to formulate assertions about individuals, i.e. instances or variables. This can be done by means of ***axioms***. They comprise additional properties beyond the regular structure of the sorts. Such properties are fully application specific and hence do not have a correspondent in the stereotype.

The formal definition of axioms doesn't prescribe the 'premise -> conclusion' form, which seems to favour a direction of inference and hence enforces aspects to be introduced on the inference layer. When introducing a

```
predicate causes( cause : top, effect : top ).
```

---

[1] Sorts are represented as a lattice, where the most general element named **top** is the upper bound of the whole lattice. There are standard sorts such as **string** and **numeric** which are also subsorts of the most general sort **top**.

the following axiom could be specified

```
axiom infection
    forall P : pathogen,
    exists D : disease;
    causes( cause : P, effect : D ).
```

Without going into any more detail, this may indicate that domain theories can as well be specified fully neutral of considerations about inference.

Given values or ground substitutions for a sufficient number of the variables, representing concrete facts about an individual entity (system_model), axioms allow to infer values for other variables of the individual or to establish or test truth conditions.

These and other elementary types of inferences are informally introduced in the subsequent section.

2.3.2 The Inference Layer

The inference layer consists of a small number of elementary inferences which can be classified into two disjoint subsets. For an inference to be elementary means that it infers a property of either the sort lattice or of a logic variable by a step which cannot be further subdivided on a conceptual level (as opposed to the implementation level, where a conceptual step may correspond to hundreds of lines of code). The two subclasses of elementary inferences are the terminological and the inferential ones.

Terminological inferences infer properties of the terminology represented by the sort lattice. Typical examples are to determine the set of subsorts of a given sort ('qualify') or its supersorts ('abstract'). The selection of other terminological inferences to be introduced shortly is guided by the principle to make available all those constituents that will be required in the example selected to demonstrate the properties of the task layer. The full collection can be found in [WETTER, 1992]. But before let us introduce the second basic group and illustrate the criteria of distinction.

Referential inferences determine properties that refer to concrete individuals or instances. Such inferences can be distinguished epistemologically and in their representation from terminological ones.

Referential inferences infer whether a property of an individual holds. This can be true or false. It can be derived by use of a theorem prover. Terminological inferences evaluate conventions. Conventions concern e.g. the set of concepts, which are subsumed by another concept. Such a set may be empty, but the inference to determine it cannot become true or false, it may provide a set which may be empty or non-empty. It operates on the data structure of the sorts, as opposed to the operations of referential inferences on substitution list for logic variables.

2.3.2.1 Terminological Inferences

One of the most obviously required basic inferences is to introduce names to store instances of sorts. For this purpose we use

```
allocate( <sort_name>, <Variable_name> )
```

e.g.

```
allocate( system_model, S )
```

in terms of the stereotype 'assessment'.

A typical use of allocated variable names in all kinds of diagnostic problem solving is that the user supplies a concrete instance or case and its characteristics and the system arrives at a decision about the instance such as more precise description, assessment etc..

A purely terminological inference is the inverse of both allocate and instantiate:

```
identify( <Variable_name>, <sort_name> )
```

determines the sort of the first argument.

For technical rather than conceptual reasons, elementary set theoretic operations upon sets of sorts are included as terminological inferences.

2.3.2.2 Referential Inferences

To express the test nature of the three referential inferences their names all end with a question mark. holds?(goal) tests wether a property expressed in goal holds, i.e. can be proved for certain values substituted for variables. holds? can obviously be represented by a usual call to a theorem prover. If such a call is to be determined under preconditions about variables

```
solves?( ->Var, goal )
```

can be used. Its effect is the same, if all variables in goal are known except Var. In that case a (new) substitution for Var is provided if the goal succeeds. If goal contains other unknown variables, solves? fails. Therefore, solves? can be summarized to supply the n+1 value iff n values are known.

The last referential inference

```
known?( ->Var )
```

tests whether a constant has been substituted for Var, i.e. whether a value has been found. It can be used as an inference but not as part of a domain layer. This guarantees that its operation cannot interfere with the one of a theorem prover.

2.3.3 The Task Layer

These and some other inferences are provided to supply the elements to be embedded in the control operations on the task layer: referential ones offer conditions that can be tested by if-then-else constructs. Terminological ones supply index sets for loops. These two notions supply the base inventory on the task layer.

Since most elements have been introduced, we provide one full example first and will address some additional issues afterwards.

```
assessment( System_model, DCF, decision_( System_model ) )
<-
determine_indicator( System_model, Setof_known_indicator );
USEALL_SORT Indicator IN Setof_known_indicator;
   allocate( Indicator, Indicator );
   IF holds?( Indicator_( System_model, Indicator ) ) THEN
      dissect( Indicator, comparison_, Setof_comparison );
      USEALL_SORT comparison IN Setof_comparison;
         allocate( comparison, Comparison );
         dissect( comparison, reference_, Setof_reference );
         USEALL_SORT reference IN Setof_reference;
            allocate( reference, Reference );
            compute( Indicator, Reference, Difference );
            IF known?( ->Difference ) THEN
               assert( judgement( arg1 : Indicator, arg2 : Reference, diff : Difference ) );
            END_IF
         END_USEALL_SORT
      END_USEALL_SORT
   END_IF
END_USEALL_SORT
USE_AXIOMS list_of_decision_class_definition( DCF )
   IF holds?( decision_class_definition( candidate : System_model, result : Decision_class ) );
      THEN assert( decision_( System_model ) = Decision )
   END_IF
END_USE_AXIOMS.
```

The overall structure of the task assessment allows the following observations:

- We find three loops USEALL_SORT ... END_USEALL_SORT embedded within each other and a bifurcation IF ... THEN as the essential control expressions.
- The elementary inferences allocate, holds?, dissect, and known? occur as goal expressions, represented by their respective inference heads and parameters.
- We find further task heads determine_indicator, select_known, and compute; the subtasks that these goals represent must still be introduced.
- assert will turn out to play a special role.

In a task an inference is represented by an inference head, which consists of its name and parameters. Non-elementary inferences - represented by task heads - can be used in any place where an inference head is used, as long as referential subtasks are taken as substitutes for referential inferences and equally for terminological ones.
assert and USE_AXIOMS add a formula to a given knowledge base resp. load a temporary knowledge base.

3 The FORKADS-Tool

In our project of formalizing and operationalizing KADS constructs we developed a modelling tool for conceptual models which generates operational code. This tool consists of two components:

- a graphical modelling component GCONMOD (cf. [HILDEBRAND, 1991])
- a generator component LCONMOD producing Prolog code (cf. [BÜRSNER, 1991])

In order to support the process of writing conceptual models in FORKADS we added to LCONMOD a graphical user interface GCONMOD. The resulting FORKADS-tool supports the knowledge modelling process of the analysis phase within the development process of knowledge based systems. The result of working with the FORKADS-tool is an *executable conceptual model* which can fulfill three functions in his role as an executable formal specification of the artefact to be built:

- prototype function
- final document of the analysis phase
- input document for subsequent design

3.1 Realization

LCONMOD

The development of LCONMOD was characterized by two competing aims:

- suitability of the language constructs for formalizing KADS interpretation models and conceptual models of real knowledge based applications
- executability of the language constructs

As our own and also VAN HARMELENS investigations (cf. [HARMELEN, VAN, 1992]) show, a *closed* formalization of the KADS layers requires very complex language means. We chose first order predicate logic as the basic language of FORKADS, because denotational and operational semantics of first order predicate logic are pretty well explored. Consequently formulations in the language can specifically be chosen such as to achieve decidability or at least enumeration of the consequences (semi-decidability).

In detail the basic language uses

- first order predicate logic
- three variations:
  - sorts (to induce a type concept of objects)
  - procedural attachment (e.g. to import results from complex mathematical computations in C functions)
  - superimposed procedural control of the logical inferences and procedure calls

The language constructs of FORKADS which are described in chapter 2 represent a mixture of logical and procedural elements. That is why the three variations of first order predicate logic are mandatory. The implementation of LCONMOD has been done in Quintus Prolog™[1] and C. For the implementaion of order sorted logic we used the inference machine of the knowledge representation language L_{LILOG} (cf. [BOLLINGER and PLETAT, 1991]) which is also implemented in Quintus Prolog.

The graphical modelling component GCONMOD is based on LCONMOD in such a manner that each user input on the graphical interface, e.g. the definition of an axiom, effects a generation instruction, e.g. the allocation of an axiom in the knowledge base. In other words, each action on the graphical user interface, like editing graphically or writing in formulars, is realized by a respective activity in the underlying software component LCONMOD. Handling the visualized formal language FORKADS demands several subgroups of activities from the underlying software:

- allocating respectively defining objects of the three KADS layers
- browsing previously allocated objects, their properties, and relations to other objects
- changing and deleting allocated objects
- invoking the theorem prover
- handling intermediate results
- realizing the effects of success and fail (e.g. adequate messages for the user in the case of syntactic and semantic faults)

GCONMOD

The graphical user interface GCONMOD is implemented in C and AT&T C++ with AIX™[2] as operating system. This implementation was supported by the following tools: OSF/Motif™[3], the graphical editor EDGE (cf. [NEWBERY-PAULISCH and TICHY, 1990]), and

---

1 trademark of the Quintus Computer Systems Corp.
2 trademark of the International Business Machines Corp.
3 trademark of the Open Software Foundation, Inc.

the compiler compiler YACC. GCONMOD is an autonomous process communicating with LCONMOD via interprocess communication.

The graphical interface consists of the following components:

- a main window
- a formular editor
- a graphical editor
- a graphical browser
- a textual knowledge base

The *main window* offers an overview over the functionality of the tool. With the help of a menu bar you have access to most functions. A control bar shows status information. Beside its function as starting window, the main window visualizes certain sort relations graphically.

The *formular editor* allows to allocate objects of the three KADS layers by editing the information in a dedicated formular. The *graphical editor* for allocating objects in diagrams can be used instead.

Dependencies of the knowledge contents as for example sort heterarchy or call trees of tasks can be visualized in the *graphical browser*. Both the graphical editor and the graphical browser apply EDGE layout algorithms for relieving users from doing manual layout and enabling visualization of objects which have either been edited in formulars or in the graphical editor.

The *textual knowledge base* is a simple data base containing objects of the three KADS layers in textual form. LCONMOD stores the knowledge in a form which is optimized for theorem proving. Since it would be very difficult to restore the user input from the optimized form, we prefered twofold storage of the same data.

An *internal representation form* serves as interface between the different components of GCONMOD. There exist three representation forms of an object:

- *text* for storage in the textual knowledge base
- *formular* for object definition respectively presentation of the object in the formular editor
- *diagram* for object definition in the graphical editor respectively object visualization in the graphical browser

All these representation forms can be re-established from the *internal representation form*. For the transformation of the textual into the internal representation form a procedure is used which was generated by YACC. The formal language FORKADS served as input grammar in the generation process. Also the predicate calls, which are submitted to the Prolog process (LCONMOD), are generated from the *internal representation form*. Such predicate calls cause for example the allocation or modification of a knowledge base object.

The sort models of the stereotypes are available in GCONMOD and can be loaded in a workspace for inspection or concrete use by means of *role_of*-declarations. There's a special function which facilitates the connection of stereotype elements with domain elements. Stereotype elements can be used as building blocks, they can be modified or rejected after

studying their procedural effects. I.e. stereotypes serve not only as mental tools for analysis but are also technical tools for studying procedural effects of data interpretations.

4 Discussion

4.1 Context of the Presented Research Activities

In our project of formalizing and operationalizing KADS constructs we aimed at a methodology that contributes to the systematic and principled production of knowledge based components of software systems. The suggested methodology should be applicable in a non-academic environment. We've still done a lot of work in consolidating the formal foundation of FORKADS (cf. [WETTER, 1992]). However, in order to verify its applicability FORKADS should now undergo some practical tests in non-academic software development environments. In parallel, further stereotypes have to be investigated and a more efficient version of the FORKADS-tool has to be implemented. Though enhancing the performance of the tool it will remain a tool for knowledge engineers and not (mainly) for domain experts. Always care must be taken not to lose too much of the modelling power of KADS when developing performance tools.

Since its first publication of an earlier version[1] the work has stimulated a research direction[2] and also influenced the formalization attempts in the KADS consortium[3]. It has helped to provide the ground for further development of the theory of knowledge acquisition[4].

4.2 Limitations and Uncovered Problems

Coverage of the FORKADS language

The precision offered by FORKADS and its tool does have its price. As expressive as the language has been designed, it is still very limited in its capacity. This does not concern the area of application such as banking, medicine, manufacturing etc. but the 'qualities' of knowledge. As to the domain structures there exist no means to deal with vagueness, such as probability or fuzziness. Time is not an explicit notion and would have to be emulated by the existing means. Corresponding restrictions hold for the inferences which directly correspond to the linguistic means offered for the static relations. Control is organized fully sequentially using loop and bifurcation. No means are available for parallel and coordinated processing of subtasks, with mutual reasoning about intermediate results of other processes.

The severely limited library of stereotypes only covers analytic problem solving. Configuration problems are within the scope of the language, but construction tasks require constructive inferences, which are principally viable, too, but not available at present. Application projects have not yet reached a state that is worth reporting experiences.

---

1 cf. [WETTER, 1990]

2 cf. [ANGELE, FENSEL, and STUDER, 1991; JONKER and SPEE, 1992; GREBOVAL and KASSEL, 1992; GAPPA and POECK, 1992]

3 through personal communication with FRANK VAN HARMELEN

4 cf. [WETTER and WOODWARD, 1990]

Activities supported

A complete methodology - in the American sense of collection of methods - must cover the full process from requirements to implementation and maintenance. The presented results respect the need for integration in such a full process but focus on knowledge acquisition and specification.

Concretely, support is supplied for

- the specification of models of expertise,
- the execution of specified models, and
- the use of such models for design.

The first two aims primarily address rational treatment of 'knowledge' per se, whereas the third aim addresses the integration of knowledge based aspects with conventional aspects of software systems. Together they realize the transition from a problem to be solved to a first precise formulation of the solution. These early phases are regarded as the most critical ones because of their inevitable 'contamination' with real world informality ([ITZFELD, 1989; TWINE, 1989]).

4.3 Other Approaches of Formalizing Models of Expertise

KADS can be seen as the presently widest spread modelling approach to knowledge acquistion. Therefore approaches to be compared with FORKADS are other attempts to formalize KADS. Before starting the comparison we recall some of the main characteristics of KADS.

Although KADS comprises a number of partly divergent currents, it seems fair to say that the KADS conceptual model (or as suggested in the newer terminology "model of expertise" [WIELINGA, SCHREIBER, and BREUKER, 1992]) is a model to make sense and the design model is a model to build a system, and that they are both not the system. Hence users of both models are humans. The conceptual model is a mental tool for the knowledge engineer, partly for the domain expert, but definitely, too for the designer. Further mental tools consist for example of interpretation models and design elements or models of cooperation.

When adhering at most expressive language means for building conceptual models, theories for full conceptual models can be aimed at. $(ML)^2$ (cf. [HARMELEN, VAN and BALDER, 1992]) is the most advanced of such approaches. However, $(ML)^2$ does not aim at full operationalization of conceptual models. It is not intended, either, to entirely replace informal models. Respecting the fact that $(ML)^2$ does not aim at operationalization, we compare FORKADS with AIDE (cf. [GREBOVAL and KASSEL, 1992]).

The **AIDE** approach tries to closely link human use and operationality of models. It distinguishes process and domain aspects and elaborates them differently from KADS. Furthermore it shows the disadvantage of being sparsely described what the structure of the languages is concerned. The strategy of AIDE differs essentially in having one "conceptual" language for the human to make sense and a compiler ("translator") into a language to be executable in a LISP environment. We find a ***structure preserving automatic transformation*** in a place

where KADS and FORKADS prescribe a ***structure preserving design***. It is important to note that GREBOVAL and KASSEL make evident the necessary condition that the human perceivable language must be a formal one. FORKADS takes the position that this is also necessary for structure preserving design (which has been argued in considerable detail in [WETTER and NÜSE, 1992]), whereas KADS claims that design can maintain the contents of a conceptual model without the conceptual model being formal[1].

There are further approaches that aim at operationalization, e.g. VITAL (cf. [JONKER and SPEE, 1992]) and OMOS (cf. [LINSTER, 1992]). Some contributions to formalizing KADS primarily aim at supplying operational *efficient* systems (e.g. MODEL-K, cf. [KARBACH et al., 1991]). Others put emphasis on relating elements of the representation language to formalisms known from software engineering and information system design (e.g. KARL, cf. [ANGELE et al., 1991]). In this field FORKADS claims to be balanced between high expressiveness within logic formalisms, executability, and human comprehensiveness, but does not aim at being an efficient target system.

Acknowledgements

We want to thank THOMAS HILDEBRAND for contributing the GCONMOD software. He has started from a rough specification and has gone through all stages of refining it and arriving at the full operational code. Thank you as well to WOLFRAM SCHMIDT, who has considerably contributed to our understanding and the present status of interpretation models.

[1] The term 'structure preserving design' has even been used for the transition from an informal KADS notation to a design model. But when accepting strong criteria such as isomorphy this cannot be the case between an informal and a formal language.

References

ANGELE, J., FENSEL, D., LANDES, D., and STUDER, R. (1991): KARL: An executable language for the conceptual model. In *Proceedings of the 6th Knowledge Acquisition for Knowledge-Based Systems Workshop KAW'91.* Banff, Canada. Oktober 6-11, 1991. P. 1/1-1/20.

ANGELE, J., FENSEL, D., and STUDER, R. (1991): Applying Software Engineering Methods and Techniques to Knowledge Engineering. *Univ. of Karlsruhe: Research Report* **204**. September 1990.

BOLLINGER, T. and PLETAT, U. (1991): The LILOG Knowledge Representation System. IBM Germany. Research Center. *IWBS Report* **156**. 1991. ISSN 0938-1864.

BRAUER, W., REISIG, W., and ROZENBERG, G. (1987): Petri nets: applications and relationships to other models of concurrency. In *APN 86. LNCS* vol. 254-255. Springer.

BÜRSNER, S. (1991): Ein KADS-orientiertes Knowledge Engineering Tool, implementiert in Variationen der Prädikatenlogik. *Master's Thesis in Medical Informatics.* University of Heidelberg. September 1991.

BRACHMANN, R. and SCHMOLZE, J. (1985): An overview of the KL-ONE knowledge representation system. *Cognitive Science* (9) 171-216. 1985.

GAPPA, U. and POECK, K. (1992): Common Ground and Differences of the KADS and Strong-Problem-Solving-Shell Approach. In WETTER, TH. et al. (Eds.). *Current Developments in Knowledge Acquisition: EKAW92. 6th European Knowledge Acquisition Workshop.* Berlin: Springer Lecture Notes in Artificial Intelligence 599. May 1992.

GREBOVAL, C. and KASSEL, G. (1992): An approach to operationalize Conceptual Models: The Shell AIDE. In WETTER, TH. et al. (Eds.). *Current Developments in Knowledge Acquisition: EKAW92. 6th European Knowledge Acquisition Workshop.* Berlin: Springer Lecture Notes in Artificial Intelligence 599. May 1992.

HARMELEN, VAN F. and BALDER, J. (1992): $(ML)^2$: A Formal Language for KADS Models of Expertise. *Knowledge Acquisition* (1992) **4**(1). P. 127-161.

HILDEBRAND, TH. (1991): Entwurf und Realisierung eines Werkzeugs zur benutzerfreundlichen Manipulation von KADS-Sprachkonstrukten. *Master's Thesis in Medical Informatics.* University of Heidelberg. December 1991.

ITZFELD, W. (1989): The GRASPIN Project and its CASE Environment Prototypes. *Final Review.* Gesellschaft für Mathematik und Datenverarbeitung (GMD). Institut für Systemtechnik. Sankt Augustin. September 1989.

JONKER, W. and SPEE, J.W. (1992): Yet another formalization of KADS Conceptual Models. In WETTER, TH. et al. (Eds.). *Current Developments in Knowledge Acquisition: EKAW92. 6th European Knowledge Acquisition Workshop.* Berlin: Springer Lecture Notes in Artificial Intelligence 599. May 1992.

KARBACH, W., VOß, A., SCHUCKEY, R., and DROUVEN, U. (1991): MODEL-K: Prototyping at the Knowledge Level. Proceedings of the 11th International Workshop Expert Systems & Their Applications. Avignon. 1991.

KLINKER, G., BOYD, C., DONG, D., MAIMANN, J., MCDERMOTT, J., and SCHNELBACH, R. (1988): Building Expert Systems with KNACK. BOOSE, J. and GAINES, B. (Eds.). *Proc. 3rd Knowledge Acquisition for Knowledge-Based Systems Workshop*. Banff (Canada). November 1988.

LINSTER, M. (1989): Integrating Acquisition, Representation and Interpretation. BOOSE, J., GAINES, B., and GANASCIA, J.G. (Eds.). *Proc. EKAW89 3rd European Workshop on Knowledge Acquisition*. Paris. July 1989. ISBN 2-90367763-8.

LINSTER, M. (1992): Linking modeling to make sense and modeling to implement systems in an operational modeling environment. In WETTER, TH. et al. (Eds.). *Current Developments in Knowledge Acquisition: EKAW92. 6th European Knowledge Acquisition Workshop*. Berlin: Springer Lecture Notes in Artificial Intelligence 599. May 1992.

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. *Int. J. Man-Machine Studies* **26** (1987). 105-121.

NEWBERY-PAULISCH, F. and TICHY, W.F. (1990): EDGE: An Extendible Graph Editor. In *Software-Practice and Experience*. Vol. 20(S1). P. 63-88.

O'NEIL, M., GLOWINSKI, A., and FOX, J. (1989): A Symbolic Theory of Decision-Making Applied to Several Medical Tasks. HUNTER, J., COOKSON, J., and WYATT, J. (Eds.). *Proc. AIME89, 2nd European Conference on Artificial Intelligence in Medicine*. London. Aug. 1989. Springer Lect. Notes Med. Inf. **38** (1989).

PLETAT, U. (1990): Knowledge Representation in LILOG. IBM Germany. Research Center. *IWBS Report 90* and *Lecture Notes in Artificial Intelligence*. Berlin: Springer. 1989.

PORTER, D. (1992): Reusable Analysis and Design Components for Knowledge-Based System Development. In WETTER, TH. et al. (Eds.). *Current Developments in Knowledge Acquisition: EKAW92. 6th European Knowledge Acquisition Workshop*. Berlin: Springer Lecture Notes in Artificial Intelligence 599. May 1992.

SCHREIBER, G., BREUKER, J., BREDEWEG, B., and WIELINGA, B.J. (1988): On Problems with the Knowledge Level Perspective. In BOOSE, J., GAINES, B. (Eds.), *Proceedings Banff'90 Knowledge Acquisition Workshop*. P. 30-1 to 30-14. University of Calgary. SRDG Publications. Also in *AISB 1991*.

TWINE, S. (1989): A Model for the Knowledge Analysis Process. *EKAW89. Third European Workshop on Knowledge Acquisition for Knowledge Based Systems*. Paris. July 1989.

WETTER, TH. (1990): First Order Logic Foundation of the KADS Conceptual Model. WIELINGA, B., BOOSE, J., GAINES, B., SCHREIBER, G., VAN SOMEREN, M. (Eds.). *Current Trends in Knowledge Acquisition*. Amsterdam: IOS Press (1990).

WETTER, TH. and WOODWARD, B. (1990): *Towards a Theoretical Framework for Knowledge Acquisition*. BOOSE, J., GAINES, B. (Eds.). *Proc. 5th AAAI Knowledge Acquisition for Knowledge Based-Systems Workshop*. Banff (Canada). Nov. 1990. Univ. of Calgary.

WETTER, TH. and SCHMIDT, W. (1991): Formalization of KADS Interpretation Models. In STEELS, L., SMITH, B. (Eds.). *AISB91. Artificial Intelligence and Simulation of Behaviour*. Springer. London 1991.

WETTER, TH. and NÜSE, R. (1992): Use of Natural Language for Knowledge Acquisition - Strategies to Cope with Semantic and Pragmatic Variation. To appear in IBM Journal Research Development.

WETTER, TH. (1992): Foundations of an Operational Language for KADS Conceptual Models. Manuscript in preparation.

WIELINGA, B.J., BREDEWEG, B., and BREUKER, J.A. (1987): Knowledge Acquisition for Expert Systems. NOSSUM, T. (Ed.). *Advanced Topics in Artificial Intelligence. Lecture Notes in Artificial Intelligence* **345**. Berlin 1987 (Springer). P. 96-124.

WIELINGA, B.J., SCHREIBER, A.TH., and BREUKER, J.A. (1992): KADS: A Modelling Approach to Knowledge Engineering. *Knowledge Acquisition* (1992) **4**(1). P. 5-53.

XIE, S., DUMARESQ, D.F., and WINNE, P.H. (1988): PRED: An Interface Development Tool for Editing Primitives in Knowledge Representation Languages. BOOSE, J. AND GAINES, B. (Eds.). *Proc. 3rd Knowledge Acquisition for Knowledge-Based Systems Workshop*. Banff (Canada). Nov. 1988.

Inferential Model and Inferential Modelling Technique: A Systematic Technique for Knowledge Analysis

© Christine W. Chan

Dept. of Computer Science
University of Regina
Regina, Sask
Canada S4S 0A2
email: chan@cs.uregina.ca

Abstract

This paper proposes the Inferential Model for facilitating the conceptual modelling phase of the expert system development process. The Model consists primarily of a classification of knowledge types expressed in the most general mathematical terms. We also suggest a systematic technique of knowledge analysis which makes use of the Inferential Model. The technique is called Inferential Modelling Technique. It is a top-down technique which can be used in conjunction with bottom-up techniques like content analysis for analysing elicited expertise in a domain unfamiliar to a knowledge engineer. Some results from using the Inferential Modelling Technique for analysing knowledge in the legal domain of negligence are also presented.

1 Introduction

Knowledge acquisition (KA) is a general name given to the process of eliciting, acquiring, and representing knowledge consisting of descriptions, relationships, and procedures in a specialized domain of interest. KA is considered a major bottleneck in the efficient development of expert systems (see for example [3]). The process of KA can be refined into the three stages of knowledge elicitation, analysis, and interpretation [2]. Recent work in this area have suggested the need to focus on individual stages within KA rather than tackle the entire process [1]. Our objective in this paper is to focus on the analysis phase of the knowledge acquisition process, and suggest a top-down model, called the Inferential Model, to facilitate knowledge analysis.

As mentioned, there are three identifiable phases within the KA process [2]. The first step of knowledge elicitation is concerned with obtaining information from an expert, which is accomplished by means of various psychological techniques such as structured or unstructured interviews, verbal protocol analysis, multidimensional scaling, conceptual sorting, participant or unobtrusive observation, introspection, etc. Machine induction has also been used in knowledge acquisition (see for example [14]); [13] contains some representative work done in this area. [6, 8, 18, 22] provide good assessments and comparisons of the various KA techniques.

The second step of knowledge analysis is concerned with how to make sense of the data collected in the first step. The third step of knowledge representation is concerned with how to express the analysed knowledge in an understandable and usable form which enhances communication between the expert and the knowledge engineer (KE) for purposes of confirmation and eventual implementation.

We focus on the analysis phase of the KA process because of a number of reasons. First, we believe this is essential to the KA as well as expert system development processes. Data and information on a domain can usually be obtained from manuals, textbooks, experts, and even users. But unless we know how to use the data, they are practically useless. In other words, data and information have to be converted into knowledge, and this conversion process takes place during the phase of knowledge analysis. Since effectiveness of the expert system to be produced depends directly on the knowledge that is incorporated, knowledge analysis is crucial to the eventual success of the system.

Secondly, the analysis step is the most difficult among the three stages in the KA process because, as A.L. Kidd pointed out, the assumption that "a magical one-to-one correspondence" between the expert's verbal comments and the real items of knowledge in his or her head is misleading. Rather, the knowledge must be *inferred* from the verbal data, and a formal, top-down approach for guiding that translation from verbal comments to expert knowledge is crucial to building successful systems:

> It is not necessary to develop a complete formal theory of any problem for this approach to be useful. At the least, a weak theory should help in classifying some of the different types of knowledge necessary to solve a particular problem and thus facilitate the process of

> analysing and interpreting data from knowledge elicitation interview and protocols. Without any such theory, it is rather like the vain hope that a room full of monkeys let loose on a batch of typewriters will eventually come up with the works of Shakespeare...[12].

A weak theory would provide a classification scheme for organizing and making sense of the abundant data collected during the elicitation stage of the KA process. The Inferential Model proposed in this paper constitutes such a "weak theory".

The third reason for our focus on the analysis stage is that there is a real lack of research on this crucial phase of the KA process. As mentioned above, there are many well-documented elicitation or acquisition techniques such as repertory grid analysis, protocol analysis, interviews, introspection, machine induction, etc. By comparison, little work has been done in developing techniques for converting data to knowledge. Conceptual modelling, as exemplified by the KADS methodology, is one way to approach this conversion, and there is a real need for support in conceptual modelling in the field of knowledge acquisition. In a review of knowledge acquisition methodologies for first generation expert systems, I.M. Neale noted the lack of research on this essential phase of the KA process saying: "the construction of this conceptual model is the least mature stage in manual knowledge engineering methodologies" [16]. Similarly, G. Smart contended that the real need in veridical or truthful knowledge acquisition is for "support in epistemology" [19].

The objective of this paper is to tackle this crucial phase in the KA process by proposing a generic or domain independent classification of knowledge types which would facilitate the analysis stage as well as provide guidelines for the elicitation of knowledge. We call the new model of knowledge analysis the Inferential Model, which functions as a "a conceptual map" for identifying the items of knowledge from the elicited expertise and serves as a framework for organizing the analysed knowledge. Based on the Model, we derive a technique of knowledge analysis called the Inferential Modelling Technique.

This paper proceeds as follows: section 2 describes the Inferential Model; section 3 outlines the Inferential Modelling Technique and how it can be used in conjunction with bottom-up techniques for analysis of knowledge, section 4 presents some results from application of the technique to analysis of knowledge in a legal domain, section 5 concludes the paper by suggesting some strengths and weaknesses of the technique in its present version.

2 The Inferential Model

Our model consists of the same four levels as the KADS interpretation model: strategy, task, inference, and domain [4]. However, the contents of the domain and inference levels differ from the KADS model. Specifically, while the KADS interpretation models do not sufficiently represent the domain level, the Inferential Model extends upon the KADS model and derives its power from detailed characterization of the domain objects and relations.

In the Inferential Model, the strategy and task levels constitute the dynamic

component and the inference and domain levels the static component. The static component consists of the domain primitives and their inter-relations, while the dynamic component describes a variety of tasks which manipulate the domain entities in order to accomplish objectives, and rationale for those tasks. The model is represented diagrammatically in figure 1 and characterized formally:

1. *Domain level*: This involves three components [9]:

   (a) objects (O): A set of observable domain objects posited by the expert as a useful ontology of the domain, each domain object consists of:

   - A set of observable property types or propositions (P) on each object, defined as a set of functions (F) which take input values and return output values;

   (b) values (V): A set of sets of symbolic, boolean, real, or integer values;

   (c) relations (R): A set of partially ordered relations on O. The relations among objects can conceptually become a set of second level objects, which has associated with it the three components of V, P, and R. In addition, a strength indicator S may be associated with each relation to represent the relative significance of the relation according to some criterion C. Theoretically, the process of building higher level propositions can continue indefinitely. But a relation which does not relate observable domain objects (O) but unobservable, also called conceptual, theoretical, abstract, or inferred objects (O_f) belongs to the inference level.

2. *Inference Level*: This involves the two considerations of inference types and strength of inferences:

   (a) Inference types: This level consists of a set of inference structures to define different types of inferences which relate a set of observable objects to a set of unobservable objects, or relate one set of unobservable objects with another set of unobservable objects. An inference structure consists of inference relations. Let R_f be a set of relations defined over a set of unobservable or conceptual objects O_f with a set of observable objects O, and $R_{f'}$ be a set of relations defined over a set of unobservable objects O_f with another set of unobservable objects $O_{f'}$.

   In both cases, some property essentially associated to the particular relation in question is preserved. A property is preserved by an inference relation if and only if the relation between two objects guarantees the second object has the property when the first one has it [9]. The property of truth is preserved in classical logic. Other meta-linguistic properties that may be preserved include coherence, relevance, necessity, normality and abnormality of function inputs and outputs, etc. [11, p.294][10]. Jennings explained the notion of inference as follows:

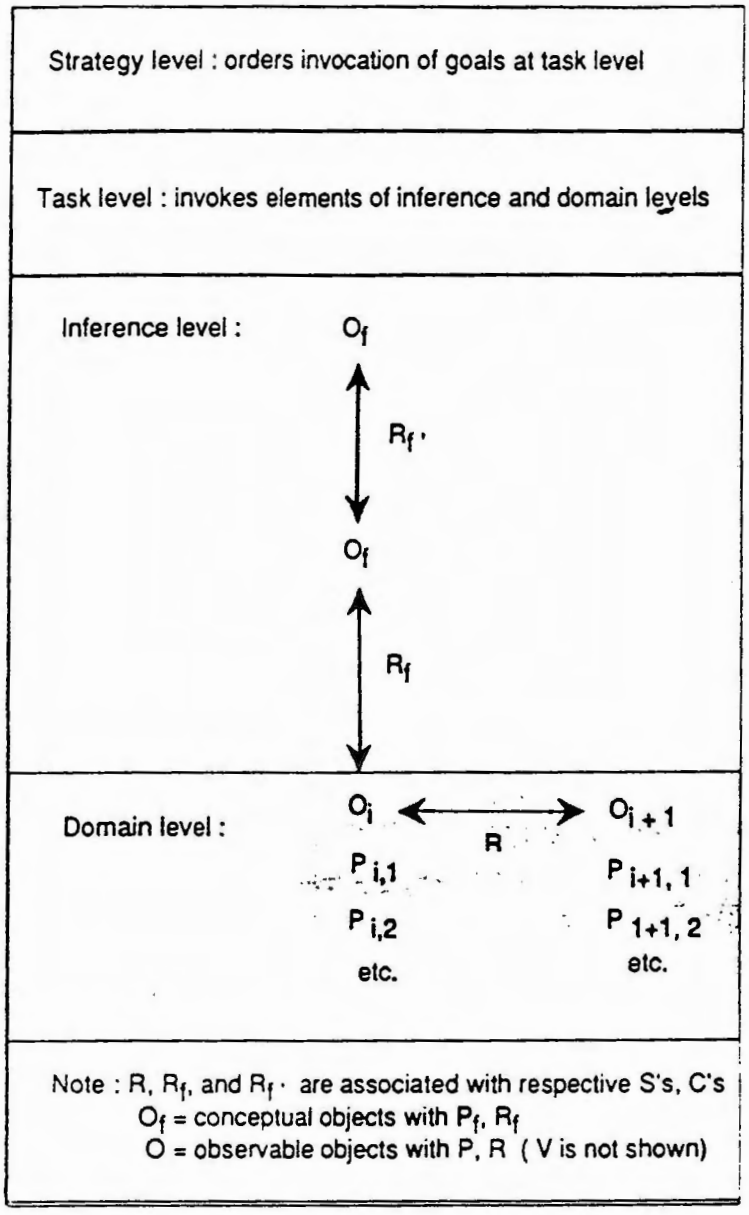


Figure 1: The Inferential Model

> Inference is essentially a procedure by which a set of sentences (for example, the set of one's beliefs) having some set of metalinguistic properties may be extended to a larger set of sentences, in such a way that that larger set is guaranteed to have the same properties. By all means, we may regard truth as one of the properties to be preserved, but what other properties are to be preserved must depend upon the requirements of particular applications. The partial corollary of the recognition of such a project is that a system of inferential principles must be tailored to the properties which are to be kept [10, p.6].

This feature of property preservation is useful for semantic validation of inferences. If the second inferred object does not have the property of the first object, and the inference relation is known to be property preserving, then an invalid inference has been drawn and the second inferred object is necessarily invalid. The distinction between O and O_f, P and P_f, and R and R_f lies in the difference between observable domain objects, their properties, and relations among them on the one hand, and unobservable or conceptual domain objects, their properties, and relations among them on the other. It is hypothesised that inferences operate on both types of objects, with the conceptual objects and inferences being the primary ingredient of expertise.

(b) Strength of inference: A strength indicator is associated with each inference relation within a structure to represent the relative inferential significance of the relation according to some criterion such as credibility, validity, importance, likelihood of occurrence, etc. For example, the certainty factor is a strength indicator that represents the inferential reliability of a relation. The choice of criterion depends on the domain; in the domain of troubleshooting an electrical device for example, frequency of occurence was deemed important [7]. The inferential relations can be partially ordered according to the criterion and the strength indicators. It is possible that within a single domain, different criteria are relevant; hence, different partially ordered sets of inference relations may be defined, which are themselves partially ordered. Corresponding to the two different types of inference relations, R_f and $R_{f'}$, the strength indicators and associated criteria are denoted as S_f, $S_{f'}$, and C_f, $C_{f'}$.

3. *Task Level.* A task is a set of related activities, the completion of which satisfies a goal. The set of activities involve procedures which activate domain and inference objects and relations, and describe how static entities of the two lower levels are used. Without this level of knowledge, system functions cannot be implemented.

   The task knowledge can be conceptualized as a set of task structures. Each structure is a network defining a method whereby O, P, R, O_f, P_f, R_f and $R_{f'}$ of the domain and inference levels are invoked to accomplish an objective. The task structures may consist of substructures so that a task objective is

accomplished by coordinating a number of subgoals. A task objective may belong to several task modalities. For example, in the problem solving modality, a task structure describes a problem solving method; whereas in the tutorial modality, a task structure may describe an explanation. A task may define a breadth-first search over a set of O's, P's, R's, O_f's, P_f's, R_f's or $R_{f'}$'s for retrieval. This structure may be invoked to fulfill the task objective of solving a specific problem or for providing an explanation. Some examples of task objectives include:

- solve a problem,
- provide an explanation,
- retrieve from a database or knowledge base some information,
- provide a solution, etc.

4. *Strategy Level.* Strategic knowledge is knowledge used by the expert to decide what action to perform next. In the Inferential Model, strategic knowledge includes only knowledge which affects the tasks addressed by the system and not that which may impact the outside world. Strategic knowledge specifies the means for achieving a task, and consists of heuristics or high level assumptions which may be derived from the nature of the problem domain or the task. It constitutes the underlying reasons for the adoption of a particular method whereby the domain and inference objects and relations are invoked. For example, in the design application, strategic knowledge consists of user specifications. These considerations often act as criteria or constraints for partially ordering objects at the other three levels. In other words, they guide invocation of goals and subgoals at the task level, as well as objects and relations of the inference and domain levels.

3 Inferential Modelling Technique (IMT)

The Inferential Modelling Technique (IMT) is a systematic technique of knowledge analysis in which the Inferential Model functions as a template for classifying and organizing the units of knowledge embedded in the verbal or textual data elicited from experts. The IMT provides top down guidance on the knowledge types that are required for problem solving and can be used in conjunction with bottom up techniques such as content analysis, verbal protocol analysis, and discourse analysis. Bottom up techniques do not provide guidelines on the types of knowledge that are required for problem solving in a domain. What often happens is that the types of knowledge implicit in a problem are considered domain dependent. Therefore, the individual knowledge engineer (KE) confronted with the task of analysing expertise is left to his or her propensity for learning about a domain as fast as possible or to his or her intuitions. For example, one effort at analysing the knowledge constructs involved in the design of business expert systems found only three relevant types: criteria, relationships, and equivocal situations or boundary of the domain [1]. The Inferential Model suggests a richer categorization scheme which includes domain and conceptual objects, properties, and their values, domain and inference relations, strength factors, criteria, task structures, and strategic knowledge. This explicit list

of knowledge types serves as a checklist, and can be used in conjunction with one of the bottom-up techniques for knowledge analysis mentioned above. Let us examine how we can use content analysis together with the IMT. The following brief synopsis of content analysis is taken from [1].

Content analysis is primarily a data filtering methodology that allows data obtained by some written, verbal, or visual communication to be transformed into a representation that is conducive to subsequent analysis. Basically, it describes a transformation process that reduces a massive amount of textual data into much fewer content categories. This data filtering process enables a researcher to extract the constructs that are of interest in an objective and systematic manner. There are two basic kinds of components extracted from textual material in content analysis:

1. recording units, which may be words, phrases, and paragraphs etc., and
2. context units, which is the largest body of context that may be examined in characterizing a recording unit.

The latter is usually needed for understanding the semantics of the former. But obviously the former is easier to identify.

The process of identifying recording units is called unitization, which is important because subsequent inferences depend upon the quality of this process. Unitization is to be undertaken simultaneously with a categorization scheme that describes the categories of interest in a domain. Currently this categorization scheme is considered domain specific:

> Categories provide a principled and objective way to obtain a successively finer refinement of domain knowledge. *The specific categories to be used in a particular situation depend upon the nature of the domain being examined* [1, p.430].

The Inferential Model suggests a generic or domain independent categorization of knowledge types. The IMT is a procedure which facilitates the development of the "specific categories" for a given domain by presenting the KE with a template of knowledge types and a sequence of steps whereby the elicited units can be classified; it consists of the following steps:

1. specify the physical objects in the domain,
2. specify the properties of objects identified in (1),
3. specify the values of the properties identified in (2), or,
4. define the properties as functions or equations,
5. specify the relations associated with objects and properties identified in (1) and (2) as functions or equations,
6. specify the partial order of the relations identified in (5) in terms of strength factors and criteria associated with the relations,

7. specify the inference relations derived from objects and properties identified in (1) and (2),

8. specify the partial order of the inference relations identified in (7) in terms of strength factors and criteria associated with the relations,

9. specify the tasks in the problem,

10. decompose the tasks identified in (9) into inference structures or subtasks (which invoke units identified in steps (1), (2), (5), and (7)),

11. specify the partial order of the inference and subtask structures identified in (10) in terms of strength factors and criteria,

12. specify strategic knowledge in the domain,

13. specify how strategic knowledge identified in (12) is related to task and inference structures specified in (9) and (10),

14. return to step (1) until the specification of knowledge types is satisfactory to both the expert and KE.

A key feature of the unitization process compatible with the application of IMT is that it involves *iterative refinement* of the unitization scheme. Iteration is also an intrinsic characteristic of the IMT (as indicated by step (14) of the procedure), and the iterative process should eventually produce an explicit classification and organization of the knowledge units identified through content analysis. When both the KE and expert are reasonably satisfied that the conceptual model that emerges represents the problem solving expertise, they terminate the process. During the iterative analysis, it is important to try to classify the identified units as best one can at the time, knowing that a first cut can always be modified during subsequent acquisition and analysis sessions. The purpose of having the Inferential Model is to facilitate this classification process by posing a checklist or categorization of knowledge types and render explicit the classification of a knowledge datum as early as possible in the knowledge acquisition process. An explicit classification, even if initially "incorrect", poses as an issue to be clarified between the KE and the expert and constitutes a step towards conceptual modelling of a problem domain.

The cyclic iteration between the elicitation and analysis stages serves to clarify in the minds of both the expert and the KE the organization of knowledge in the domain. When the analysis phase has been completed to the satisfaction of both parties, the knowledge representation phase can begin. The knowledge constructs can be mapped directly onto some chosen implementational mechanism; alternatively, they can be translated into an intermediate logical representational scheme such as pseudo-English if-then statements before being converted into code.

4 Application of the IMT for analysis of knowledge in the domain of the Remoteness Advisor

We apply the Inferential Modelling Technique to analyse knowledge in the legal domain of negligence and discuss its utility in clarifying the types of knowledge implicit in the problem. Briefly, a central issue in the law of negligence is the problem of *remoteness*, i.e. how far down the chain of causality could a person be liable for a negligent act so that a damage would be recoverable. Negligence usually involves the creation of risk of harm to someone. The injuries of the primary subject of the risk are generally not very remote; the remoteness issue arises when other persons suffer a loss as a result of the negligence to the primary subject [20, p.127].

The Remoteness Advisor is an expert system which aids lawyers determine if remoteness is involved in a case. The objectives of the Remoteness Advisor are: (1) to establish whether negligence and remoteness is involved in a given case, and (2) if so, to categorize the negligent act that has led to the damage, (3) also if so, to retrieve all prior court cases which could be used as precedents in court.

The Remoteness Advisor is directed at the third objective. The extent to which the system is able to identify precedent cases that are as similar (or dissimilar) as possible to the one that is to be presented in court is one of the key measures of its value to lawyers. However, similarity in the domain seems to be an elusive concept. Therefore, the task becomes one of finding a scheme to classify a given situation in order to be able to compare it to prior cases. This is an unstructured task and the scheme to be constructed to a large extent depends on the expert's judgement on what is a good match between the given and prior cases and what factors, concrete and conceptual, have to be taken into consideration to make such a determination. The author obtained data on the domain through unobtrusive observation [23] and taped protocol of the KA sessions between the expert and KE. Since the KE was a lawyer by training, he shared the expert's language and familarity with the set of cases relevant to the domain. During the KA sessions, they worked at formulating a top-level categorization scheme of all the cases. The resultant structure is shown in figure 2 and in IMT terms, can be conceptualized as a controlling inference structure. Some sample units of classified knowledge within the structure are presented as follows:

1. *Domain level*:

   (a) objects (O): Observable domain objects include the identified parameters of interest in relevant cases, e.g., weapons (including firearms), machinery, explosives, fire, inflammable substances, toxic substances, electricity, radioactivity, animals, planes, vessels, and animal powered vehicles. The main property of interest for each object is its existence, i.e., whether the object in fact plays a role in a specific case. More detailed properties then depend on the particular role each object assumes in the case.

   (b) values (V): In the legal domain, V consists of sets of boolean or symbolic values.

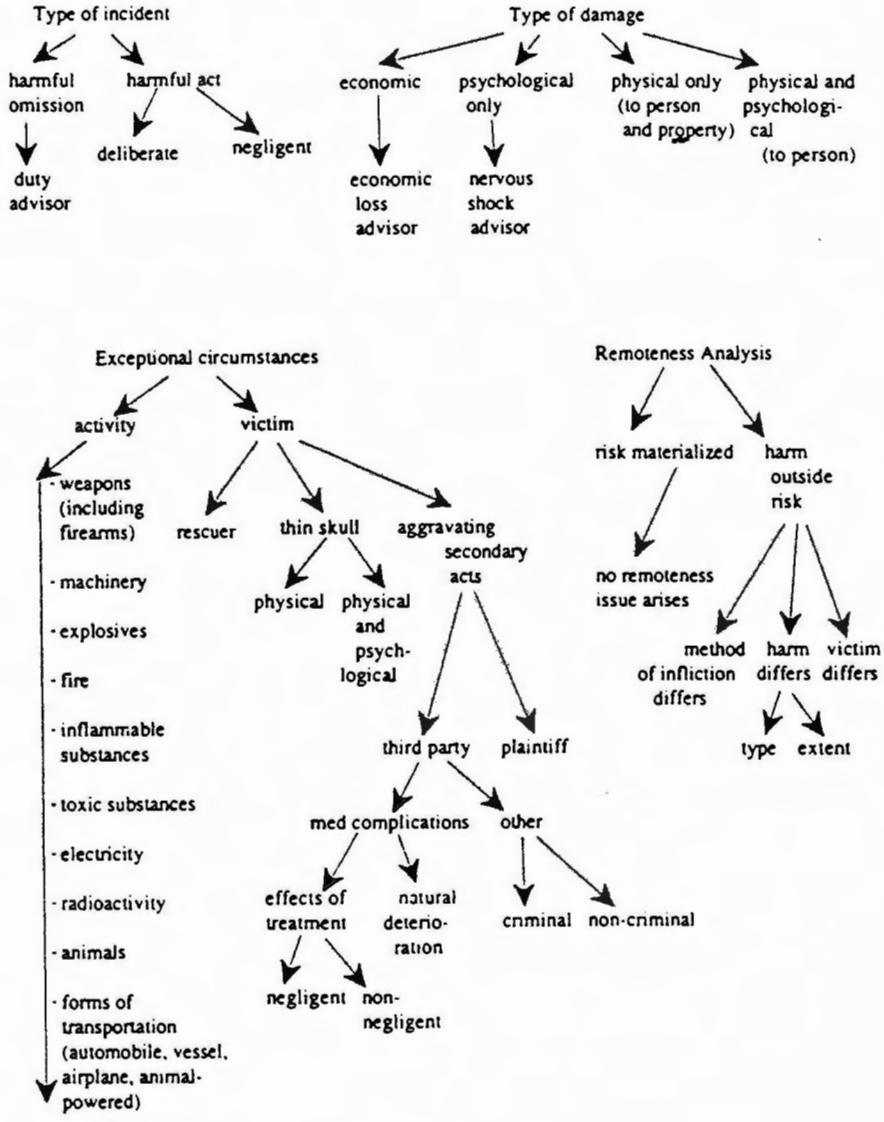


Figure 2: Structural representation of expertise in the EPSILON project

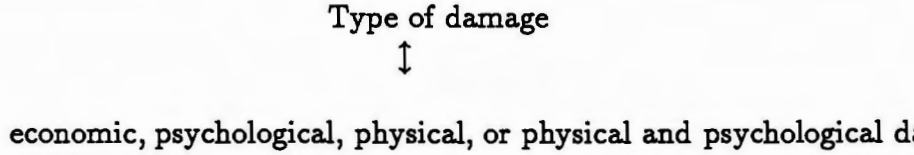


Figure 3: Two level inference structure

(c) relations (R): R among observable objects within a case may indicate their interdependencies. For example, a causal relation may exist between the objects of explosives and fire in that the former causes the latter. Or, R among observable objects across different cases may be relations of similarity or dissimilarity. For example, if two cases both involve explosives, the relation of similarity connecting the two objects of explosives across the cases indicates how the two objects play similar roles in the respective cases, and the relation of dissimilarity connecting them describes how they differ. Strength factors associated with the relations were not considered during the acquisition.

2. *Inference Level*:
A sample inference is shown in figure 3 and discussed as follows. The type of damage may be actual economic, psychological, physical (to person or property), and physical and psychological (usually to person). The observable objects (O's) are the economic, psychological, physical, or physical and psychological damages, and the unobservable or conceptual object (O_f) is the type of damage. The inference relation connecting the four actual damages to their type can be denoted as R_f and is indicated in the figure as a bidirectional arrow. In the downward direction, R_f is a specialization relation, and in the upward direction, R_f is a generalization relation. The property preserved in this inference relation is 'damage involved.' Inferential significance of the relations were not considered.

3. *Task Level*:
The function of the Remoteness Advisor is to aid lawyers in classifying a case and retrieving similar or contrary cases. Problem solving thus involves identifying key features or parameters of the cases which would adequately classify them. In other words, the Advisor is a classification system [4].

4. *Strategy Level*:
In constructing the system, the expert decided that concrete parameters such as the means of transportation (e.g. whether a plane, automobile, ship, or animal is involved in a case) form the basis of the first categorization scheme. When this concrete level of filters fails to classify a case, then a second level categorization scheme which was formulated from more abstract or conceptual

considerations such as the plaintiff's relationship to the risk (e.g. whether the plaintiff is a rescuer or prone to injury) was invoked. This two level categorization scheme was deemed sufficient to cover all cases involving remoteness.

5 Conclusion

In this paper we have presented the Inferential Model which is in fact a domain neutral classification of the types of knowledge that constitute expertise. The Model represents a "weak theory" for guiding the analysis of elicited knowledge because it only suggests to the KE how the different types of knowledge *may be* related. Actual organization of knowledge items for a specific application is likely to be highly domain dependent. By itself, the Inferential Model provides a classification or categorization of knowledge types in a domain in the most general mathematical terms. When it is used in conjunction with an objective and systematic knowledge extraction technique such as content analysis, the IMT is a powerful tool for making sense of data on a domain unfamiliar to a KE.

The Inferential Model describes the types of knowledge at the epistemic level, independent of implementational mechanisms to which the knowledge constructs would eventually be translated. In other words, analysing the elicited data using the Inferential Model ensures the analysis is at the knowledge level [17] and not influenced by implementational considerations. This avoids any "representational mismatch" [5] between the acquired knowledge and the knowledge representational mechanism eventually employed. Motta et al. described the advantages of having a specification of knowledge constructs which the Inferential Model presents succinctly as follows:

> A prerequisite to the extraction of knowledge from a knowledge source is a specification of the precise form of the objects that constitute knowledge. Such a formalization of knowledge constructs is useful for a variety of reasons. First, it provides the knowledge engineer with a template which he/she can use when analysing descriptions of problem solving behaviour, regardless of the manner in which these descriptions were generated. Second, cognisance of knowledge constructs can serve as an input into the process of information acquisition, thus reducing the dichotomy between information and knowledge and shortening the knowledge acquisition cycle. Finally, the use of such a formalization makes the knowledge extraction process independent of its eventual representation, thereby reducing the possibility of a representation mismatch. The specification generated by knowledge extraction can then be further refined and structured to implement the expert system using any representation scheme and /or software development tool [15].

The Inferential Model is deficient in its present version in that it does not account for temporal data and reasoning; hence, extensions to the Model is necessary before it can be applied to process domains. The design of the Model is still evolving. With

further applications to diverse domains, the Model will be refined. We will report on the refinements in the future.

Acknowledgements

The co-operation and insights of the expert and KE who constructed the Remoteness Advisor are gratefully acknowledged. I would also like to thank the anonymous reviewers for their constructive comments. Finally, I would like to acknowledge the generous support of the Graduate Fellowship of the British Columbia Advanced Systems Institute, the Centre for Systems Science Graduate Scholarship, the Ebco/Epic Scholarship for Expert Systems, and the Research Stipend and Graduate Fellowships of Simon Fraser University.

References

[1] Agarwal, R. and Tanniru, M.R. "Knowledge extraction using content analysis", *Knowledge Acquisition* (1991) **3** (4), 421-441.

[2] Bell, J., Hardiman R.J., "The third role - the naturalistic knowledge engineer", in Diaper, D. (Ed.), *Knowledge Engineering : Principles, Techniques, and Applications*, Chichester : Ellis Horwood Ltd., 1989.

[3] Breuker, J.A. and Wielinga, B.J., "Knowledge acquisition as modelling expertise: the KADS methodology", *Proceedings of the first European Knowledge Acquisition Workshop*, Reading, Great Britain, 1987.

[4] Breuker, J.A. and Wielinga, B.J., "Use of models in the interpretation of verbal data", in Kidd, A. (ed.), *Knowledge acquisition for expert systems : a practical handbook*, New York : Plenum Press, 1987.

[5] Buchanan, B.G. et al., "Constructing an expert system", in Hayes-Roth, R., Waterman, D.A., and Lenat, D.B. (Eds), *Building Expert Systems*, London: Addison-Wesley, 1983.

[6] Cordingley, E.S., "Knowledge elicitation techniques for knowledge-based systems", in Diaper, D. (Ed.), *Knowledge Elicitation : Principles, Techniques and Applications*, Chichester: Ellis Horwood Ltd., 1989.

[7] Davis, R., "Diagnosis via causal reasoning : paths of interaction and the locality principle", in *Proceedings of the National Conference on Artificial Intelligence* Washington, D.C. (1983), 88-94.

[8] Greenwell, M., *Knowledge Engineering for Expert Systems*, Chichester : Ellis Horwood Ltd., 1988.

[9] Jennings, R.E., "Notes on an inferential model for automated diagnosis", unpublished notes, 1990.

[10] Jennings, R.E., Dowad, M., and Apostoli, P., "Natural Inference", unpublished paper, 1989.

[11] Jennings, R.E. and Johnson, D.K., "Paradox-tolerant logic", *Logique and Analyse*, 1984.

[12] Kidd, A.L. (ed.), *Knowledge Acquisition for Expert Systems: a Practical Handbook*, New York: Plenum Press, 1987.

[13] Michalski, R.S., Carbonell, J.G., and Mitchell, T.M. (Eds.), *Machine Learning: An Artificial Intelligence Approach Vol.1 and 2*, California: Morgan Kauffmann Publishers, Inc., 1986.

[14] Michalski, R.S., and Chilausky, R.L., "Learning by being told and learning from examples: an experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis", *International Journal of Policy Analysis and Information Systems*, Vol. 4, No. 2, 1980, 125-161.

[15] Motta, E., Rajan, T., and Eisenstadt, M., "Knowledge acquisition as a process of model refinement", *Knowledge Acquisition*, **2**(1), 21-49.

[16] Neale, I.M., "First generation expert systems: a review of knowledge acquisition methodologies", *The Knowledge Engineering Review*, **3**(2), 1988, 105-145.

[17] Newell, A., "The Knowledge Level", *The AI Magazine*, Summer 1981, 1-20.

[18] Slatter, P.E., *Building Expert Systems: Cognitive Emulation*, Chichester : Ellis Horwood Ltd., 1987.

[19] Smart, G., "How to make expert systems...and stay in business", London, U.K. 2-4 June 1987.

[20] Smith, J.C., *Liability and Negligence*, Carswell Legal Publications, 1984.

[21] Tong, X., Karbach, W., and VoB, A., "Filling in the knowledge acquisition gap: via KADS' models of expertise to Zdest-2's expert systems", *Proceedings of the European Knowledge Acquisition Workshop for Knowledge-Based Systems (EKAW '88)*, Gesellschaft fur Methematik und Datenverarbeitung, MbH, GMD, 1988.

[22] Welbank, M., "Perspectives on Knowledge Acquisition", British Telecom, 1986.

[23] Webb, E.J., Campbell, D.T., Schwartz, R.D., and Sechrest, L., *Unobtrusive measure: nonreactive research in social sciences*, Chicago: Rand McNally, 1966.

Graph-Grammar Assistance for Modeling of Decisions

John W. Egar, Angel R. Puerta, and Mark A. Musen
Knowledge Systems Laboratory
Stanford University
Stanford, CA 94305-5479
email: {egar, puerta, musen}@camis.stanford.edu

Abstract

One of the most difficult aspects of modeling complex dilemmas in decision-analytic terms is composing a diagram of relevance relations from a set of domain concepts. The concepts, as presented to the modeler, generally have no inherent order, yet they have characteristic positions and typical roles in a semantic network model. We have been using a graph-grammar production system to express such inherent interrelationships among terms. We have found that this graph-grammar system facilitates the modeling of dilemmas.

1 Modeling of Decisions

Where judgments are made under appreciable uncertainty, and where stakes are high, decision makers may need help in weighing risks and benefits involved in important decisions. Decision theory (von Neumann and Morgenstern, 1944) can provide such help. Unfortunately, constructing decision-analytic models is a difficult task, even for trained analysts.

We have implemented a *graph-grammar production system* that constructs decision-analytic models automatically from unordered lists of terms. A *graph grammar* is a system of replacement rules that operates on graphs, rather than on the strings on which traditional context-free grammars operate. Our present underlying graph representation is an extension to the *influence-diagram* notation (Howard and Matheson, 1984), where nodes represent probabilistic variables, and arcs represent either probabilistic dependencies between the variables or information available at the time of a decision. Our system is an automated modeling tool that accepts an unordered list of terms that together describe a dilemma, and produces an initial influence diagram. Since the person entering the terms does not need to be a renowned expert, we shall refer to that person as the *practitioner* for the remainder of this paper.

The decisions that we have been modeling involve selecting an optimal plan for medical intervention. The plan may consist of tests, treatments, and waiting periods, and is tailored to individual patients, each of whom has her own particular set of priorities and her individual probability distributions for the various chance outcomes. Since the individual patient must make a decision for their particular circumstances, we need to tailor a model to fit each dilemma so that the model can properly guide the patient, or some health-care worker acting for the patient, toward a normative decision.

Some decision analysts who use influence diagrams become facile with this representation as they grow familiar with typical interactions. These typical interactions can help to guide practitioners who have not studied a great number of decision-analysis problems, and who find the modeling task daunting. In many ways, this situation parallels the development of computer programming: One of the reasons that we can write computer programs today that are larger and more sophisticated than those developed 40 years ago is that we have available higher-level programming languages. Researchers have formulated sufficient dilemmas as probabilistic networks to allow us to discern recurrent patterns. We can begin to make use of these patterns as higher-level building blocks in modeling decisions.

2 GRAPH GRAMMARS

A graph grammar consists of a set of production rules that dictates how a graph can be transformed and rewritten. These production rules are quite different from the productions used in rule-based expert systems: Graph-grammar rules can specify a wide range of contexts for which they are applicable, and can describe different graph manipulations for those different contexts. A graph grammar specifies a language over a set of symbols, the members of which are elements of a graph. We have found graph grammars to offer an expressive and concise way to represent prototypical forms for modeling dilemmas. Also, graph grammars can provide high-level abstractions that help users to manage complexities.

Over the past 20 years, a plethora of formalisms has been developed to describe rewriting procedures for graphs. These formalisms include algebraic graph grammars, array grammars, collage grammars, edge-label–controlled grammars, expression grammars, graphic sequential rewriting systems, hyperedge-replacement grammars, map grammars, neighborhood-controlled embedding grammars, node-label–controlled grammars, picture-layout grammars, plex grammars, precedence graph grammars, relation grammars, shape grammars, and web grammars (Ehrig et al., 1987; Ehrig et al., 1991; Feder, 1971; Pavlidis, 1972). The particular formalism that we use is a modification of Göttler's operational graph grammars (Barthelmann, 1991).

Each production rule in a grammar describes a *legal graph manipulation.* Using Göttler's formalism, we write these productions as graphs divided into four *regions* (Figure 1), which partition the vertices into four sets: those in the left region, V_{L}; those in the right region, V_{R}; those in the *determinate* region below, V_{B}; and those in the *indeterminate* region above, V_{A}. The two sets V_{A} and V_{B} are referred to as the *embedding part.* The graph manipulation described by such a production is as follows: Find nodes matching the left region, V_{L}, and replace them with nodes matching the right region, V_{R}. The procedure consists of these four steps:

1. Find a region of the host graph where the vertices and edges match the vertices and edges of the determinate (V_{B}) and left (V_{L}) regions of the production (Figure 2a).

2. Find zero or more edges that match edges between the left (V_{L}) and indeterminate (V_{A}) regions of the production. (Because edges between V_{A} and V_{B} require special treatment, our grammar does not include productions with such edges.)

3. Remove from the host graph those vertices that matched vertices within V_{L} (Figure 2b).

4. Add to the host graph new vertices and edges that correspond to those within the right region (V_{R}) of the production, and add to the graph edges that correspond to those connecting the embedding part (from V_{B} and the matched portion of V_{A}) of the production to the right region (V_{R}) (Figure 2c).

Our graph-grammar production system requires a set of such replacement rules. Often, where there is simple node replacement with no constraints on the embedding environment, we shall abbreviate such productions with string-grammar productions.

3 INFLUENCE DIAGRAMS

In the influence-diagram notation, there are only four types of nodes: chance nodes, decision nodes, deterministic nodes, and the utility node (Figure 3). Decision nodes stand for a set of exclusive alternatives among which the decision maker must choose. A chance node represents a random variable that describes an uncertain outcome or an unknown factor. Deterministic nodes are a degenerate version of chance nodes: Deterministic nodes are computable, given the values of their parent nodes. The utility node is a special deterministic node the value of which the decision maker wishes to maximize.

An influence diagram consists of a set of these nodes in a *directed acyclic graph* (*DAG*). The arcs in this DAG are solid when their target is a chance, deterministic, or utility node. Solid arcs denote a probabilistic dependence of one variable (indicated by the target node) on another variable (the source node for the directed arc). When their target is a decision node, the arcs are dashed;

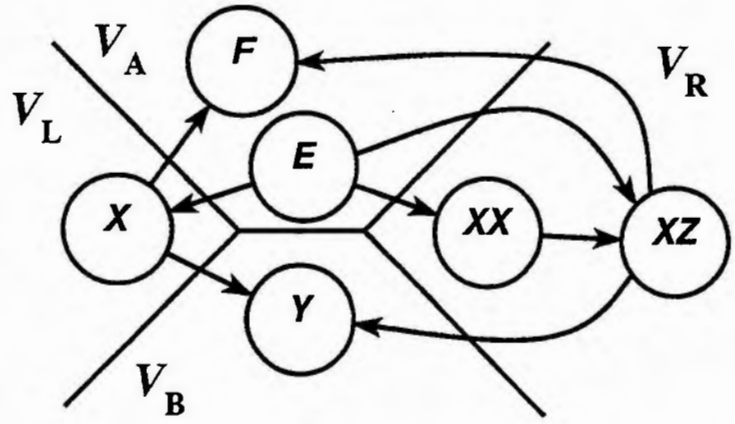


Figure 1: Sample graph-grammar production rule. The production denotes that some node of type X in the host graph is to be replaced by new instances of XX and XZ. V_L, V_A, V_B, and V_R are the four regions of a graph-grammar production rule.

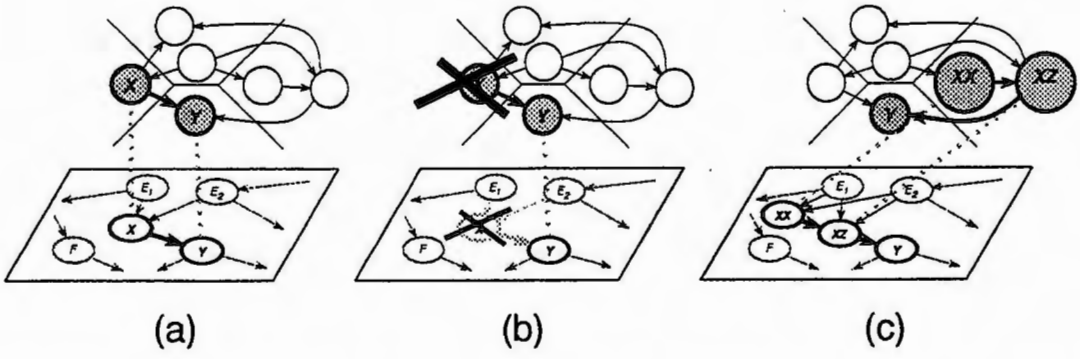


Figure 2: Sample application of the graph-grammar rule from Figure 1. (a) The first view of the host graph shows two nodes matching X and Y in the production. (b) The node matching X is removed. (c) Additional nodes XX and XZ are added.

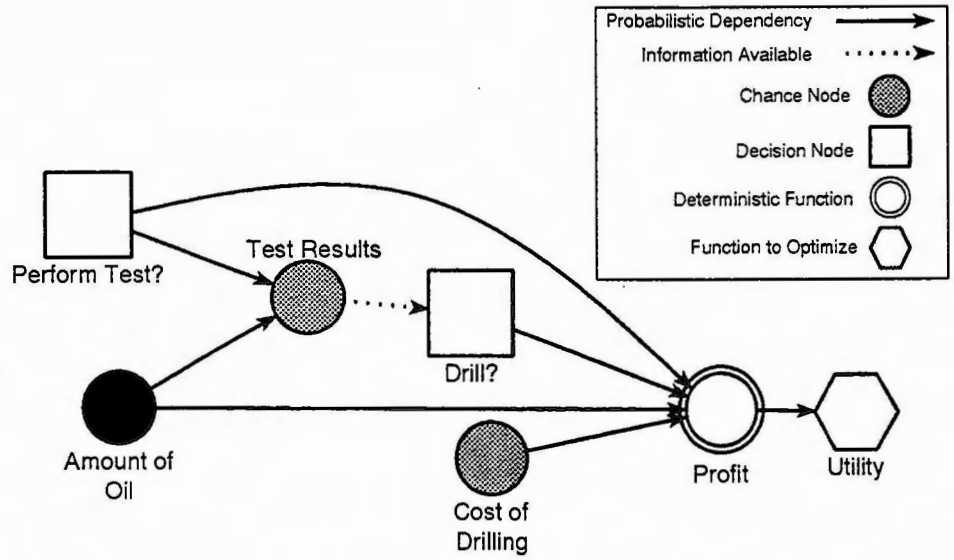


Figure 3: An influence diagram model for an oil prospector's decision. The prospector must decide whether testing and drilling are worth their costs.

they represent the availability of information (about the source node's value) at the time when the decision maker must choose among the alternatives represented by the target node.

Influence diagrams offer an explicit representation of probabilistic independence, denoted by the lack of an arc between two nodes. They also grow linearly with the number of factors being modeled.

As a simple example of an influence-diagram model, Figure 3 describes the sort of decision that a prospector might face when he has (a) the opportunity to test the soil of a potential oil reserve, and (b) the chance to drill for oil at that site. Performing the test costs a fixed amount of money, and drilling costs a variable amount of money. If he does drill for oil, he can earn an amount of money that depends on the amount of oil found at that site. His goal for this endeavor is to make as much money as possible; therefore, the value he wants to maximize is some function of the profit.

From the diagram in Figure 3, we see that the prospector's decision to perform or not to perform the test affects the test results (since they will not be available if he does not perform the test) and his overall profit (since the test costs money). His decision to drill will be made with information from the test results (if he performs the test), and the decision will affect his profit. The amount of oil at that site is an uncertain variable that is related to the test results, and to the prospector's profit. The cost of drilling is an uncertain factor necessary to compute his profit. Finally, the diagram models his utility as a function of the profit and of no other variable.

Without *assessing* the model (i.e., assigning specific probability distributions and deterministic functions to each node), we can still provide more information about the prospector's dilemma. We know that profit is directly related to the amount of oil found, and that profit is inversely related to the cost of drilling. Most relationships between variables representing domain concepts have a typical qualitative value—either proportional or inversely proportional. We can represent these qualitative relationships by labels on the probabilistic (solid) arcs: A plus sign, "+," indicates a direct relationship; a minus sign, "−," indicates an inverse relationship; a question mark, "?," indicates an unclear or non-monotonic relationship.

Also, we know that there are *asymmetries* buried in the model. Here, we mean by *asymmetries* those relations—probabilistic or informational arcs—that are valid for only particular values of certain nodes. For example, the informational arc from test results to the drilling decision makes sense only if the prospector decides to perform the test. Because these asymmetries are common, we use contingency nodes, a notation described by Fung and Shachter.[1] In this notation, we

[1]Robert M. Fung and Ross D. Shachter. Personal communication.

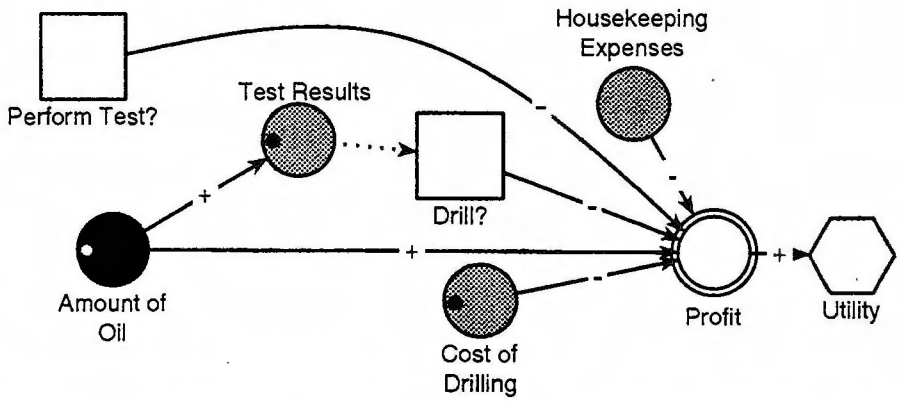


Figure 4: A model using a *quantitative contingent influence diagram* (*QCID*), representing an oil prospector's decision. Aside from the addition of "`Housekeeping Expenses`," the problem is identical to the problem described in Figure 3. However, in this notation, asymmetries—such as the pertinence of test results only when the test has been ordered—are represented explicitly with contingent nodes, which have dots on their right sides. Our use of contingent nodes makes trivial dependencies—such as the dependence of "`Test Results`" on "`Perform Test?`"—implicit. Also, the solid arcs distinguish probabilistic relations that are proportional from those that are inversely proportional. The former are labeled with "+," the latter are labeled with "−."

divide a node into several contingent nodes, each with exclusive conditions. Each contingent node is considered relevant to the rest of the diagram for only those scenarios in which its conditions are met. If a contingent node's conditions are not met, then the node and its incident arcs can be ignored. By introducing contingent nodes to the influence-diagram representation, we gain the ability to express common-sense principles that may be obvious to people modeling decision problems, but that would not otherwise be apparent in the influence diagram.

In Figure 4, we show for the prospector's dilemma a model that uses a *qualitative contingent influence diagram* (*QCID*). With QCIDs, we label the probabilistic (also called *conditioning*) arcs with the sign of their qualitative relationship. We also label each contingent node with one or more dots on its left side, each dot representing a constraint on some other variable (or set of variables) that must be satisfied for that node to be relevant to any of its successors.[2] The qualitative arcs in Figure 4 express constraints on each outcome's probability distribution. For instance, the prospector's profit is inversely related to the housekeeping expenses, and his utility is directly related to his profit.

By adding qualitative labels to the arcs and contingencies to the nodes, we obtain a model that is closer to being an assessed influence diagram, yet the model is still general enough that we do not need to redesign the model for each test, for each prospector, or for each potential drilling site. On the basis of our qualitative model, the prospector cannot decide whether he should test or drill, or whether he should be willing to invest $100,000 in the test: To arrive at such normative decisions, he must quantify his utility as a function of his profit, and he must assess the precise probability distributions for each outcome. Still, the construction of the QCID model is a major component of the overall modeling task.

Numerous inferential procedures have been developed to aid us once the diagram has been constructed (Pearl, 1986; Lauritzen and Spiegelhalter, 1988; Chavez and Cooper, 1990; Heckerman, 1991), yet there are few tools to help us construct the diagram. Other researchers have provided rough strategies for formulating decision-analytic models (Leong, 1991; Wellman et al., 1992) and have built systems that critique decision-tree models (Wellman et al., 1989). In some cases (e.g., (Hollenberg, 1984; Holtzman, 1988; Jimison, 1990)), a prefabricated model for a particular domain can be simplified according to the features of an individual decision. Still, systems that help decision

[2] In the QCID models that our production system develops, each contingent node is associated with explicit conditions on other nodes through special contingency arcs. These attributes could be represented easily in the diagram as a third type of arc, but such a notation would defeat the visual simplicity of the diagrams.

makers to formulate models for the spectrum of clinical dilemmas are not available. It has been our experience that diagrams with less than a dozen nodes can still take a trained decision analyst weeks to construct, even when the analyst is familiar with the problem domain.[3] Also, there is little that we can do to assess the validity of the model and its implicit assumptions, particularly when the chance variables are neither clearly defined nor standardized across models. Given these modeling difficulties, we are not surprised that normative decision making is not as prevalent as we would wish. Consequently, we have concentrated on automating the initial modeling of a decision problem, and on maintaining a lexicon of standard symbols that have clear denotational and referential semantics.

4 A Graph-Grammar Production System for Modeling of Medical Decisions

The context that we wish to assume for our graph grammar is as broad as possible. Still, our graph grammar for medical dilemmas reflects a few assumptions about what constitutes a dilemma. We assume that the decision maker prefers health to illness and to death, and that she wants to avoid costs, risks, disability and pain. We also assume that she subscribes to the basic axioms of decision theory (North, 1968). Certainly, these assumptions are not universally applicable; still, we believe that they are reasonable. Otherwise, our grammar presumes little about the context of the medical problem. For example, we assume that the qualitative relationship between surgery and the risks that surgery entails is positive, and that the utility of these risks is negative. Our grammar makes no commitment about many of the details of the model, because it operates on models that are not yet assessed. How a state variable is descretized, how its outcomes are ordered, what an outcome's marginal probability is, what the conditional probabilities on an outcome are, what an outcome's utility is, and how the decision maker's overall utility is modeled—all these details must be assessed after the user constructs the QCID. Although some sequences of observations and decision alternatives can be dismissed before the problem is assessed fully, most dilemmas in medical decision making involve weighing the uncertainties and the utilities of qualitatively plausible scenarios. Consequently, before the practitioner can determine which decision alternatives have the most favorable expected outcome, she will have to assess probabilities and utilities for her model. If modeling is made particularly easy, however, practitioners are more likely to develop larger and more robust models. We can design a simple diagnostic model by using rules that emphasize signs, symptoms, and diseases. Later, the user can begin to design normative plans (Wellman, 1990) by using rules that introduce costs, risks, and decisions that are sensitive to utility. Moreover, if we are concerned with decisions or values from a small area of a large QCID that we have constructed with our graph grammar, we can use well-understood operations (Shachter, 1990) to determine which nodes are relevant and need to be assessed.

4.1 A Simple Derivation

To see how our graph-grammar production system is able to generate models, we shall construct a model for a simple testing decision. In this example, the practitioner considers whether or not she should perform a test for the disease tuberculosis (TB). There is some chance that her patient has tuberculosis, and there is a treatment available that often cures tuberculosis. The terms that describe her dilemma can be extracted from this brief description: "`TB`," "`TB treatment`," and "`TB test`."

We begin our derivation with a single node: the utility nonterminal symbol. Since, according to our grammar, "`TB`" is a terminal symbol that can replace the nonterminal "`(disease)`," and diseases can be added to a model as parents of the utility node (see production (a) in Figure 6), the system adds "`TB`" (corresponding to the node in the left region of the rule) to the model, and it adds an arc from "`TB`" to the utility node. The label on this new arc is a minus sign. The system chooses the most specific rule that would incorporate one of the terms into the evolving model. (The system searches the right regions of rules for a node that could be replaced by "`TB`;" then, the

[3] Yuval Shahar. Personal communication.

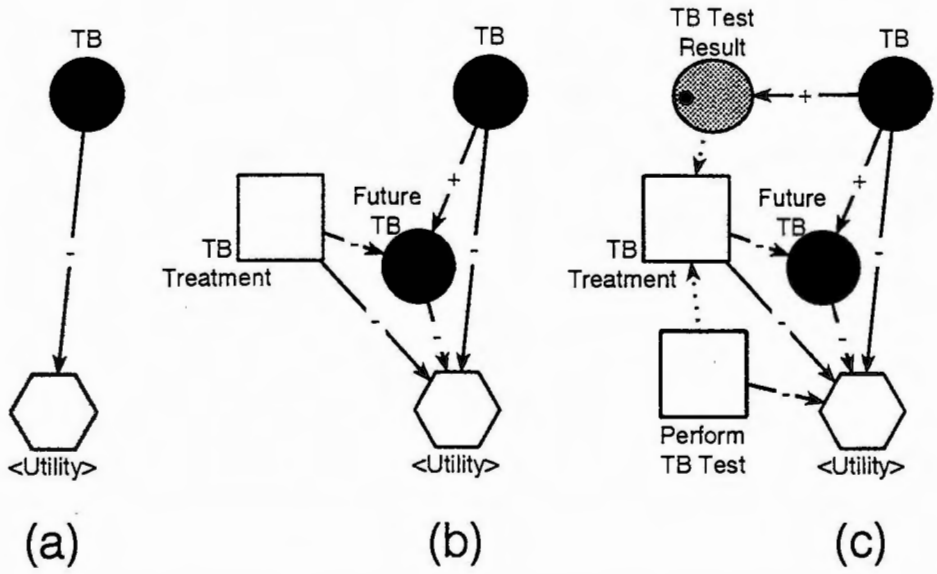


Figure 5: Derivation of a test dilemma. The practitioner for this model must decide whether to perform the TB (tuberculosis) test, and whether to administer the TB treatment. (a) Early model with the symbol for "TB" included. (b) Model with "TB" and treatment included. (c) Model with "TB," "TB treatment," and "TB Test" included. For visual clarity, we color disease nodes black and other chance nodes light grey.

system tries to match the rule's bottom region with nodes that are already in the evolving model.) The model is at the stage shown in Figure 5a.

Next, our system notes that "TB treatment" is a curative treatment, and finds the rule (production (a) in Figure 8) that might incorporate such a curative treatment into the model. In accordance with that rule, we insert nodes for the treatment and for the state of "TB" into the model. The model is at the stage shown in Figure 5b.

Finally, the system has one more terminal symbol to add to the model: "TB test." The rule for adding such a test is shown in Figure 7b. In accordance with the rule, the system adds "perform TB test" and "TB test result" to give us the model shown in Figure 5c. Our system, as implemented in Common Lisp, performs this three-step derivation without any assistance from the user.

4.2 Standard Vocabulary

Our use of a standard domain vocabulary as the acquirable knowledge representation minimizes the conceptual distance from the practitioner's description of the problem to the computational model, as recommended by Gruber (Gruber, 1989). Ideally, the vocabulary is already a standard that is used by practitioners, and that has well-defined referential semantics, which are found in some authoritative text. Also, the optimal mapping from standard terms in the vocabulary to terminal symbols in the knowledge representation is 1:1 (bijective). If the mapping is 1:N, then the vocabulary may lack the breadth needed to express computationally useful distinctions.[4] A mapping of N:1 may be due to synonymous terms in the domain argot, or may indicate that the knowledge representation is missing distinctions that domain experts have found to be useful. This latter lack of expressiveness may not present an obstacle if the types of problem solving to be performed with the model are more limited than are the types encountered by a domain expert.

Standard vocabularies also provide a degree of referential semantics, and, consequently, unambiguous communication and straightforward reuse of models by other researchers. Medical lexicographers have spent many years collecting diagnostic and procedural terms to be used in billing and in coding summaries for medical reporting. Two standard lexicons that are increasingly used

[4]Some grammars use context to resolve ambiguous mappings.

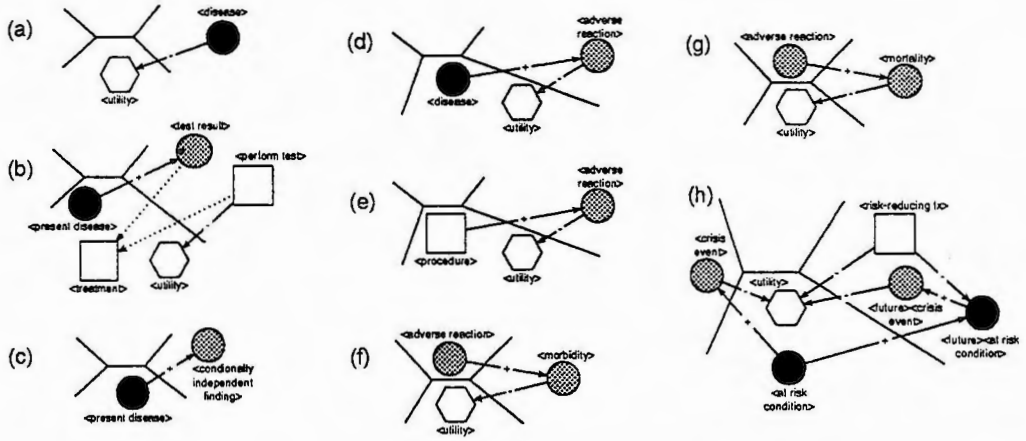


Figure 6: Sample grammar rules. Along with a simple classification of node labels, these rules can derive the example model in Figure 9.

```
1. (and (nodetype ?decision test-decision)
2.      (non-degenerate? ?decision)
3.      (member ?branch (branches ?decision))
4.      (function-value choice-action ?branch ?action)
5.      (typep ?action test)
6.      (not (null? ?action)
7.      (not (and (member ?x (downstream-nodes
                        (branch-node ?branch)))
8.                (typep ?x test-result?)
9.                (function-value
                        test-result-test ?x ?branch)
10.               (function-value some-branch
                        ?x ?branch1)
11.               (function-value get-plan
                        ?branch1 ?plan1)
12.               (member ?branch2 (branches ?x))
13.               (function-value get-plan
                        ?branch2 ?plan2)
14.               (not (plan-equal?
                        ?plan1 ?plan2)))))
```

(a)

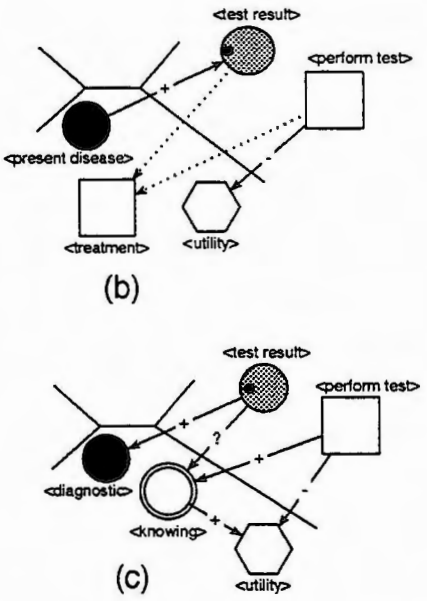


Figure 7: Representation of a clinical maxim. (a) The way that BUNYAN models the clinical rule that no test should be ordered unless a future decision depends on the test results (Wellman et al., 1989). (b) The corresponding rule from our graph grammar. (c) A rule from our grammar that allows exceptions, where there is an inherent value to knowing the test result.

in medicine are SNOMED (Coté, 1982), and CPT (Finkel, 1990). SNOMED covers diagnoses and clinical observations; CPT lists medical tests and procedures. CPT also provides detailed textual definitions for each term. These two standard vocabularies comprise most of the terminal symbols for our graph-grammar production system. More accurately, the vocabularies provide labels for symbols that are either (1) decision nodes, for tests and procedures; or (2) chance nodes, for test results, findings, and diagnoses.

SNOMED and CPT, however, are not designed for building decision-theoretic models, and they are not comprehensive. We have found that we require a few other terminal symbols to model medical dilemmas. SNOMED inadequately represents subjective symptoms (e.g., quality and severity of pain), clinical signs, and risk factors from a patient's history. SNOMED often represents different measures of severity as different terms for the same diagnosis. We find it easier to model these different SNOMED entries as outcomes of a single chance variable; hence, we delay the modeling of severities until the refinement step of knowledge acquisition. Cost, noxiousness, and other value-oriented variables are not modeled at all by CPT and SNOMED. Also, we have found that modeling dilemmas requires a notion of time, including present and future tenses as modifiers to the basic set of terminal symbols.

Because of these deficiencies in SNOMED and CPT, other researchers in our laboratory have been designing a formalism for temporal abstractions (Shahar, 1991), and have been building extensions to SNOMED that will cover the salient information contained in the medical record (Campbell and Musen, 1992). We are using these extensions, SNOMED, and CPT as the labels for all the terminal symbols in our generated models of medical dilemmas.

4.3 Nonterminal Symbols

The nonterminal symbols that we use derive from the different patterns that we see in manually composed influence diagrams; only later are these patterns given names that resemble the distinctions physicians make. For instance, from observing how treatment decisions typically fit into influence diagrams, we discerned five basic patterns (Figure 8). Only then did we give the patterns the suggestive names "`curative tx`," "`palliative tx`," "`preventive tx`," "`ablative tx`," and "`empiric tx`."[5] (The abbreviation "`tx`" stands for "treatment.") Since these names are merely labels for nonterminal placeholders, the system would function in exactly the same way had we assigned the patterns arbitrary labels instead.

Moreover, we can compare patterns for different nonterminals to arrive at a more coherent grammar. For instance, we isolated the pattern in Figure 8h, "`risk-reducing tx`," before we isolated the other treatment-decision patterns. Then, when we compared these patterns while ignoring their labels, it became obvious that "`risk-reducing tx`" was just a particular expansion of the "`curative tx`" nonterminal pattern. Such comparisons may have been obscured in a knowledge-level analysis that is less well-defined in terms of formal symbol structures than is our QCID representation. Our analysis of prototypical patterns can detect only those distinctions that emerge from the set of relations and node types defined by the underlying knowledge-representation language. Consequently, our grammar cannot construe all the physiologic, pathologic, and anatomic distinctions that are important in medicine. Nonetheless, for the restricted problem domain of medical dilemmas, we have not found these latter distinctions to be essential.

4.4 An Example Modeling Problem

To test the utility of our graph grammar for formulating decision-analytic models, we implemented a domain-independent derivation system. We then used the system to derive numerous QCID models; we also tried to derive several other models that are described in the literature (Langlotz, 1989; Jimison, 1990; Holtzman, 1988). To derive a model, we begin with a single utility node, which is in some sense the starting nonterminal, or sentence symbol, for the grammar. Then, we look at the textual description of the problem, and we select productions based on the terms in the description. When we finish listing these terms, the production system will have derived a QCID model. The

[5] These labels are actually labels for the nonterminal nodes being added by particular rules. The rules and patterns themselves have no names.

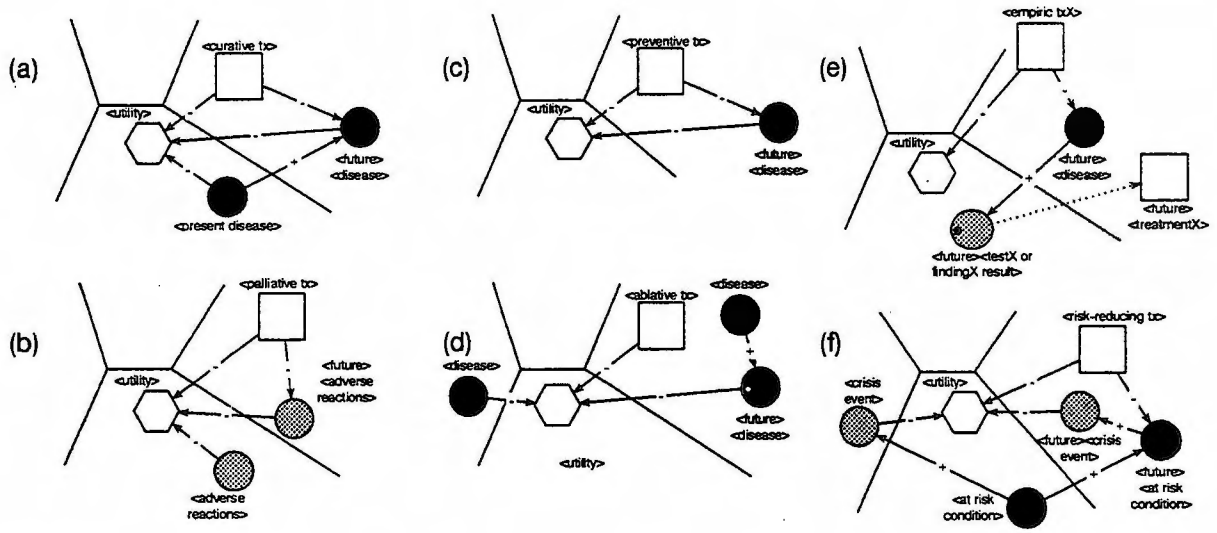


Figure 8: Five basic patterns for treatment decisions, and one derivative pattern. Shown are production rules for incorporating decisions about (a) curative, (b) palliative, (c) preventive, (d) ablative, (e) empiric, and (f) risk-reducing treatments into a model. The rule for risk-reducing treatments (f) is a special case of the rule for curative treatments (a).

decision maker can then make simplifications—some guided by graph-grammar productions, and others performed by direct inspection. At this point, the qualitative model is ready for assessment.

As an example, the following clinical problem is supplied by Wellman:[6]

> A patient with a known *history* of *coronary artery disease* (CAD) and *cerebrovascular disease* (CVD) presents with a large *abdominal aortic aneurysm* (AAA).... There is a surgical procedure to *repair the AAA*, but surgery carries the risk of operative *mortality* and *disability*. The operation is especially risky for this patient because the CAD increases the likelihood of a heart attack (MI, for *myocardial infarction*) during AAA surgery, and the CVD enhances the probability of a *stroke*. (Wellman, 1990, p. 9)

The hypothetical decision maker also considers performing two tests: *carotid arteriography,* to assess the extent of the patient's CVD, and *cardiac catheterization,* to assess the extent of his CAD. In addition to the AAA repair, she has in mind two concomitant procedures: *coronary-artery bypass-graft* (CABG) surgery, to relieve cardiac ischemia caused by the CAD, and *carotid endarterectomy,* to reduce the risk of stroke from the CVD.

Several of the productions called for by these concepts are listed in Figure 6. Additional term-subsumption productions that are easily expressed as graph-grammar rules, but that are more succinctly listed as string-grammar rules, follow:

diagnostic	→	malady | salubrity | condition
malady	→	injury | disease | at-risk condition
adverse reaction	→	noxiousness | toxicities | crisis | cost
reaction	→	adverse reaction | propitious reaction | unclear reaction | morbidities | mortalities |...
at-risk condition	→	CAD extent | CVD extent | AAA extent |...
crisis	→	MI | stroke | AAA rupture |...
test result	→	coronary-angiography result | carotid-arteriography result |...
conditionally independent finding	→	patient history | physical examination |...

[6] We italicize precisely those standard terms that our system uses to generate an initial model.

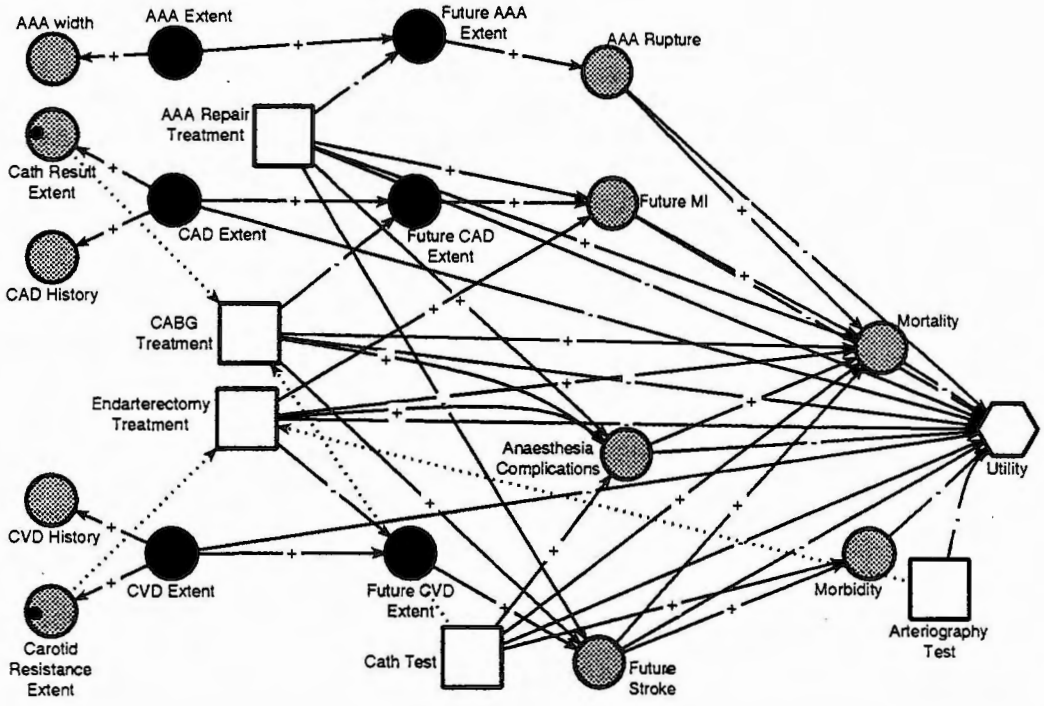


Figure 9: Sample model of a medical dilemma, derived from a graph grammar. The model is based on a case from Wellman (Wellman, 1990). In this case, a patient with coronary-artery disease (CAD) and cerebrovascular disease (CVD), presents with an abdominal aortic aneurysm (AAA). The decision maker is considering repair of this AAA, carotid endarterectomy (Endarterectomy), coronary-artery bypass-graft (CABG) surgery, carotid arteriography (Arteriography), and coronary-artery catheterization (Cath). Myocardial infarction (MI, or heart attack), stroke, AAA rupture, and complications of anaesthesia are the risks to consider.

$$
\begin{aligned}
\text{patient history} &\rightarrow \text{patient history CAD} \mid \text{patient history CVD} \mid \ldots \\
\text{decision} &\rightarrow \text{wait} \mid \text{procedure} \mid \ldots \\
\text{procedure} &\rightarrow \text{test} \mid \text{treatment} \mid \ldots \\
\text{treatment} &\rightarrow \text{curative tx} \mid \text{palliative tx} \\
&\quad \mid \text{preventive tx} \mid \text{ablative tx} \mid \text{empiric tx} \\
\text{curative tx} &\rightarrow \text{risk-reducing tx} \mid \ldots \\
\text{risk-reducing tx} &\rightarrow \text{CAD-reducing tx} \mid \text{CVD-reducing tx} \\
&\quad \mid \text{AAA repair} \mid \ldots .
\end{aligned}
$$

First, we note those diseases that are being considered. For each disease, we can add a chance node and a negative conditioning arc from that disease to the utility node, in accordance with production (a) in Figure 6. Once we have specified disease nodes in our model, we can apply productions (b) and (c) to add tests to the model. Note that these latter productions apply not only when we have diseases in our diagram, but also when we have diagnoses with positive or unknown utilities (e.g., pregnancy). In our grammar, the nonterminal node "⟨`diagnosis`⟩" is more general than is "⟨`disease`⟩," as indicated by the abbreviated grammar productions listed previously.

The practitioner can then cite other factors from the medical description of the problem. For each term that she cites, she has one or more productions that she can apply at various locations in the evolving QCID. In some situations, the production system can find obvious matches for vertices within a rule. For example, once AAA repair is entered, the "⟨`at-risk condition`⟩" node

in production (h) can be matched automatically to the "(AAA extent)" node, and the "(crisis event)" node from the production can be matched automatically to "(AAA rupture)." In other situations, the user may have to choose the location at which to apply a rule. As the user finishes applying productions for the remaining medical terms in the problem, the system constructs the diagram shown in Figure 9. Aside from the possibly superfluous arcs from crisis events to the utility node, this diagram is essentially the same as that composed by Wellman (Wellman, 1990). Our QCID knowledge representation, however, does not include the qualitative synergies employed by Wellman. We believe that the inclusion of these constraints on intercausal dependencies would be a simple extension to the grammar. It might also help us to categorize concept pairs in terms of their causal interactions.

5 Graph Grammars in Other Domains

Although our grammar is specific to medical dilemmas, the basic structure (i.e., the nonterminals) may be applicable to other domains of decision analysis. For example, prospecting for oil and mineral deposits involves diagnosis and testing decisions similar to those in medicine. The symbol "(diseaseX)" in Figure 7 merely represents an unknown that has inherent disutility, whereas the other symbols in that rule are common to decision analysis for any domain. Problems in business administration and in public planning are often fraught with uncertainties, and the gravity of decisions in these worlds often merits use of formal analysis. As in medicine, different types of patterns may emerge because of the manner in which chance and decision nodes are typically included in decision-theoretic models. We have experimented with a separate grammar for business decisions—a grammar that our system used to derive the QCID in Figure 4. However, we have insufficient experience with this sort of modeling, and we have no codified vocabulary for the business domain. Still, the basic decision-analytic patterns that emerge in our grammar do not seem idiosyncratic to medicine.

Beyond decision analysis, Göttler's operational graph-grammar formalism can be applied to other labeled, directed-graph representations. Engineers often model systems using Bond graphs (Karnopp et al., 1990), circuit diagrams, Petri nets, state-transition diagrams, entity–relationship diagrams, and PERT charts. If practitioners from engineering domains can supply a standard vocabulary, and if they can abstract prototypical patterns from their diagramming experience, they may benefit from the assistance of a graph-grammar modeling system (Göttler et al., 1991).

6 Discussion

We have observed that the models that our graph-grammar production system has constructed are essentially equivalent to the models that we and other researchers previously constructed without such a system. Since such qualitative models can take a skilled analyst weeks to compose, our system can make the task of modeling quicker and easier for decision analysts. More importantly, since most difficult decisions are not made with the help of a decision analyst, our system could be an important component in a computerized decision-support tool for practitioners. However, as promising as our results have been, we cannot claim that the equivalence of generated and previously composed models validates the models generated by our grammar. Heckerman has shown that the construction of a global knowledge map (an influence diagram with no decision or utility nodes) from the union of local knowledge maps is sound and preserves the information stated in the local maps (Heckerman, 1991). We do not have a similar justification for the models our system produces. Moreover, our grammar does not model adequately complex subgraphs of parameters that separate findings from diagnoses. To overcome this deficiency, we may combine the initial models that the grammar generates with prefabricated network models of medicine and human physiology (Beinlich et al., 1989; Shwe et al., 1991; Herskovits and Cooper, 1990). Those subgraphs of a prefabricated network that relate parameters in the initial, grammar-generated model can be incorporated into the model, thereby correcting assumptions of independence.

Another problem is that our system does not assign a complete order to the decision nodes. Wellman has defined a model for examining qualitative influence diagrams and reducing the number

of reasonable orderings for the decision nodes (Wellman, 1990). Nonetheless, since proper analysis of an influence diagram demands that the decision nodes be totally ordered, our production system will need additional information about the domain so that it can assist the practitioner in the ordering process.

We suspect other deficiencies in our grammar will emerge. Consequently, we must continue to rely on the practitioner to examine and make adjustments to the generated qualitative model. Meanwhile, we are revising the grammar to minimize any need for assistance from the user.

6.1 Refinement of the Qualitative Model

The graph-grammar system that we have developed does not yet support the assessment of probabilities and utilities, both of which are needed for the practitioner to arrive at normative decisions and plans. Many of the probabilities can be gleaned from medical databases and from assessed medical belief networks. Boolean operators and other prototypical causal interactions also have been used to assist the practitioner in assessing conditional probabilities (Rousseau, 1968; Wellman, 1990)(Pearl, 1988, pages 184–194). Other researchers have designed expert systems that help the decision maker to arrive at probability assessments (Holtzman, 1988). By using similarity networks, experts can assess probabilities for densely connected diagnostic nodes more easily and accurately than they can using a single influence diagram (Heckerman, 1991). We also suspect that we can use similarity networks to assess decision nodes that have many influential factors and exclusive alternatives.

6.2 Graph Grammars as a Modeling Language

A characteristic of our graph-grammar formalism is that it can express maxims for clinical reasoning clearly and concisely. For example, researchers in computer-assisted decision analysis developed a program, called BUNYAN, that examines a decision tree for signs of common modeling errors (Wellman et al., 1989). One of the rules that BUNYAN uses to critique a model of a medical dilemma is shown in Figure 7. The rule derives from the clinical heuristic that tests should not be ordered unless future decisions depend on their results. On the left side of Figure 7 is the top-level description that BUNYAN uses for this rule. The pattern language that it uses is defined elsewhere in the program. Moreover, the functions and variables used have suggestive labels and descriptions, but they are defined only in BUNYAN's source code. In contrast, Figure 7 also shows the corresponding rule in our graph grammar.

6.3 Translation and Assembly

The manner in which our graph-grammar production system models decisions appears to be a process akin to compilation: An unordered set of domain terms is compiled by our graph-grammar system into a diagram, with well-defined denotational semantics for computational inference. We shall refer to this perspective as the *translate-and-assemble* view for knowledge acquisition (Figure 10).

In more detail, translation and assembly represent only two steps from the more lengthy mapping from referenced objects in the world to programming structures in the computer (Figure 10). The first step, translation, is merely a direct mapping from a set of standard terms from the domain to a set of terminal symbols. The second step, assembly, consists of incorporating each of the terminal symbols given by the practitioner according to the assembly system (in our case, a graph-grammar production system). If the practitioner has access to data concerning the referents, or if she has other, prefabricated models that are relevant to the problem at hand, then she uses them to adjust the initial model during refinement. The translate-and-assemble perspective makes no commitment on how the processes of perception, enunciation, or model refinement should be conducted. So even if—as some scientists suggest (Clancey, 1991)—a particular domain expert does not have a model in mind until after he enunciates his thoughts, this cognitive deficiency does not affect the process of relating the concepts in these thoughts into a putative model. The translate-and-assemble perspective *does* suggest that, through careful formalization of the source language (domain termi-

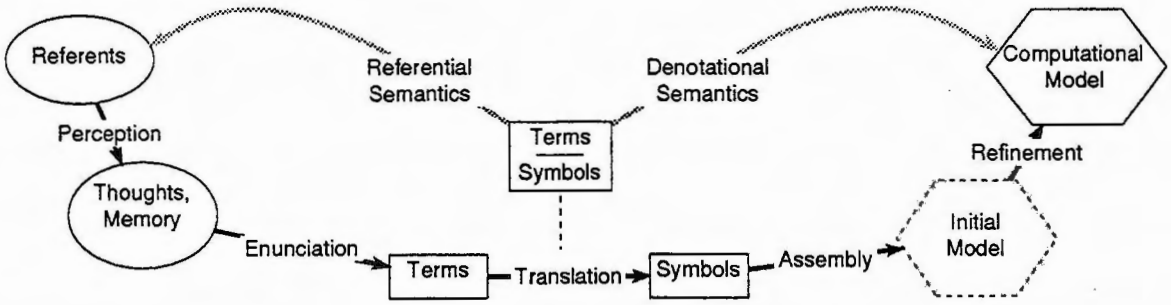


Figure 10: Transformation steps associated with the translate-and-assemble view of knowledge acquisition. Referents—objects being modeled—are transformed by the practitioner into thoughts and memories. The practitioner then enunciates these mental models as standard terms, such as "`stroke`" and "`mortality`." A simple mapping translates the terms into symbols. A modeling system generates an initial model from these symbols. The modeler corrects the initial model and adds details to create a working computational model.

nology), and of the target language (computational structures), knowledge engineers may be able to automate significant portions of formulation of tailored models for individual domains.

Bradshaw, Covington, Russo, and Boose divide automated support for knowledge acquisition into role-limiting methods, mediating knowledge representations, interviewing techniques, and analysis tools (Bradshaw et al., 1990). According to this view, we might classify our decision-theoretic framework as a role-limiting method, but the term *method* must be stretched to cover the general category of all inferences that can be made from an influence diagram. In some ways, our influence diagrams are a mediating knowledge representation, but the practitioner does not draw out the initial diagram as input to the tool, and, although the influence diagram *may* be the medium of communication during the refinement of the model, *other* representations may be used instead (see Section 6.1).

Our graph-grammar production system does not appear to fit into Bradshaw's classification of tools. The translate-and-assemble perspective provides no guidance on interviewing techniques or analysis tools: The perspective requires only that a set of terms is somehow elicited, and that the initial model is somehow refined. However, grammars may predict constructs (prototypical patterns, such as "`palliative treatment`") that may guide elicitation. Since the modeling process in our translate-and-assemble system is directed by a grammar, the tool resembles a syntax-directed editor. Unlike syntax-directed editing, however, the editing is performed automatically by the system operating on an unordered set of terms.

Researchers have found that graph grammars are a useful tool for modeling in numerous fields, including software engineering, parallel programming, database design, developmental biology, cartography, and housing construction (Ehrig et al., 1991; Hess and Mayoh, 1990). Since much of knowledge acquisition is viewed as a modeling task, knowledge-acquisition researchers can benefit from a better understanding of graph grammars.

7 Summary

We have found that our graph-grammar production system facilitates the modeling of medical dilemmas. The standard vocabulary used by the system provides a general set of medical concepts, each with clear referential semantics. The QCID knowledge representation provides a well-defined, albeit limited, denotational semantics for our decision-analytic models. Graph grammars address relationships between domain concepts other than lexical ordering; consequently, graph grammars are ideally suited for deriving models from unordered lists of terms. The grammar that we employ does not ensure that the models produced are appropriate for a practitioner's problem, but it does guide the practitioner to compose models of medical decision making that we believe are reasonable.

Acknowledgments

We are most grateful to Henrik Eriksson and Ross Shachter for their support and guidance. We thank Lyn Dupré for editing a previous draft of this paper.

We conducted this work with the support of the National Library of Medicine under grants LM-05157, LM-07033, and LM-05208, and of the Digital Equipment Corporation. Dr. Musen is recipient of National Science Foundation Young Investigator Award IRI-9257578.

A Göttler's Graph-Grammar Formalism

In the more formal terms of Barthelmann's notation, we define a labeled directed graph as comprising three sets:

1. A set of vertices (V), with labels (L_V) and a mapping ($l_V : V \rightarrow L_V$) from vertices to their labels
2. A set of permissible edge labels (L_E)
3. A set of labeled directed edges ($E \subseteq V \times V \times L_E$)

A production is a labeled directed graph, with vertices in four disjoint sets: V_{L}, V_{R}, V_{B} and V_{A}. Edges between vertices in V_{L} and V_{R} are not permitted.

A.1 Connected Vertices

A spanned subgraph, span(V', G), of the host graph G and spanning vertices V' consists of vertices ($V' \subseteq V_G$), their labels (L_{V_G}), the edges between vertices in V' ($V' \times V' \times L_{V_G} \cap E_G$), their labels ($L_{E_G}$), and the restricted labeling function $l_V(V')$. A *chain* is a sequence of edges without regard to their direction. So, a chain between v_0 and v_N exists if and only if

$$(\forall i \in \{1, \ldots, n\})(\exists m \in L_E)((v_{i-1}, v_i, m) \in E \vee (v_i, v_{i-1}, m) \in E).$$

For a given node v, and for a given set of vertices V_X, all the nodes v' that are reachable by some chain through nodes in V_X are considered *connected* to v, and this relation is denoted $v \sim v'$. We shall use the connection relation to determine which of the vertices in the indeterminate region are to be mapped to the host graph, where we shall add edges as specified in the production.

A.2 Applicability of a Production

If all vertices in the left and bottom regions of the graph-grammar production are matched to vertices in the host graph, and if the edges among the vertices in the left and bottom regions of the production are matched to corresponding edges in the host graph, then we can apply the rule. However, we must note which nodes and edges in the host graph match those in the indeterminate region and are connected, by the reflexive $\sim$ relation described in Appendix A.1, to the nodes and edges that we have already matched to the left and bottom regions of the production.

So, more formally, a production rule p is *applicable* to some subgraph of G indicated by the monomorphism $\delta : \mathrm{span}(V_{\mathrm{L,p}} \cup V_{\mathrm{B,p}}, p) \rightarrow G$ if and only if

$$(\forall v \in V_{\mathrm{L,p}})(\forall(\delta(v), v', m) \in E_G)$$
$$(\delta(v), v', m) \in \delta(E_p \cap V_{\mathrm{L,p}} \times V_{\mathrm{L,p}} \times L_E) \cup \bigcup_{C \in V_{\mathrm{A,p}}/\sim} \bigcup_{\mu \in M_\delta(C)} \mathrm{Old}(\mu)$$

and

$$(\forall v \in V_{\mathrm{L,p}})(\forall(v', \delta(v), m) \in E_G)$$
$$(v', \delta(v), m) \in \delta(E_p \cap V_{\mathrm{L,p}} \times V_{\mathrm{L,p}} \times L_E) \cup \bigcup_{C \in V_{\mathrm{A,p}}/\sim} \bigcup_{\mu \in M_\delta(C)} \mathrm{Old}(\mu),$$

where

$$\begin{aligned} M_\delta(C) &= \{\mu : \text{span}(V_{\text{L,p}} \cup V_{\text{B,p}} \cup C, p) \to G | \mu(\text{span}(V_{\text{L,p}} \cup V_{\text{B,p}})) = \delta\} \\ \text{Old}(\mu) &= \{(\delta(v), \mu(v'), m) \mid (v, v', m) \in E_p \cap V_{\text{L,p}} \times (V_{\text{A,p}} \cup V_{\text{B,p}}) \times L_E\} \\ &\quad \cup \{(\mu(v'), \delta(v), m) \mid (v', v, m) \in E_p \cap (V_{\text{A,p}} \cup V_{\text{B,p}}) \times V_{\text{L,p}} \times L_E\}. \end{aligned}$$

The set of edges $\text{Old}(\mu)$ contains those edges between the vertices to be removed and the embedding environment in the host graph. The set C represents connected vertices in V_{A}. The set of edges $M_\delta(C)$ contains those edges in the host graph that are matched to those in V_{A} according to μ—a particular (indeterminate) extension of the vertex monomorphism, δ. The edges in the set $\bigcup_{C \in V_{\text{A,p}}/\sim} \bigcup_{\mu \in M_\delta(C)} \text{Old}(\mu)$ are those edges that connect vertices to be removed with matched subgraphs in the connected environment, along with edges in that matched subgraph. So, all host-graph edges incident to the vertices that match those in V_{L}, according to δ, can be divided into

1. Edges that match edges in V_{L}
2. Edges that match edges (including matched target subgraphs) from V_{L} to the embedding environment
3. Edges that do not match edges in the production

We must distinguish the target subgraphs of the second group of edges because they are precisely those subgraphs that may need to be connected to the inserted subgraph (V_{R}) according to the production.

A.3 Effects of a Production

When a production p is applied to the host graph, there are three basic effects:

1. We remove labeled vertices matching those in V_{L}.
2. We add labeled vertices corresponding to those in V_{R}, even if similarly labeled vertices already exist in the host graph.
3. We add labeled edges among the new nodes and between the new nodes and subgraphs of the host that were connected to the removed vertices and matched according to a particular extension of δ, all according to the production.

Stated formally, the effect of applying production p on graph G at the subgraph indicated by $\delta : \text{span}(V_{\text{L,p}} \cup V_{\text{B,p}}, p) \to G$ is the graph H:

$$\begin{aligned} V_H &= V_G \setminus \delta(V_{\text{L,p}}) \uplus V_{\text{R,p}} \\ l_{V_H} &= l_{V_G}(V_G \setminus \delta(V_{\text{L,p}})) \uplus l_{V_p}(V_{\text{R,p}}) \\ E_H &= E_G \cap (V_G \setminus \delta(V_{\text{L,p}})) \times (V_G \setminus \delta(V_{\text{L,p}})) \times L_E \\ &\quad \cup E_p \cap V_{\text{R,p}} \times V_{\text{R,p}} \times L_E \\ &\quad \cup \bigcup_{C \in V_{\text{A,p}}/\sim} \bigcup_{\mu \in M_\delta(C)} \text{New}(\mu), \end{aligned}$$

where

$$\begin{aligned} M_\delta(C) &= \{\mu : \text{span}(V_{\text{L,p}} \cup V_{\text{B,p}} \cup C, p) \to G | \mu(\text{span}(V_{\text{L,p}} \cup V_{\text{B,p}})) = \delta\} \\ \text{New}(\mu) &= \{(\mu(v), v', m) | (v, v', m) \in E_P \cap (V_{\text{A,p}} \cup V_{\text{B,p}}) \times V_{\text{R,p}} \times L_E\} \\ &\quad \cup \{(v', \mu(v), m) | (v', v, m) \in E_P \cap V_{\text{R,p}} \times (V_{\text{A,p}} \cup V_{\text{B,p}}) \times L_E\} \end{aligned}$$

and where the labeling set does not change:

$$L_{V_H} = L_V$$
$$L_{E_H} = L_E$$

The edges $\bigcup_{C \in V_{A,p}/\sim} \bigcup_{\mu \in M_s(C)} \text{New}(\mu)$ are the edges that connect vertices to be added with the matched subgraph in the connected environment, along with edges in the subgraph that is matched to V_A.

B Qualitative Contingent Influence Diagrams

A *qualitative contingent influence diagram (QCID)* is a semantic network that describes a set of possible scenarios, S. For each scenario, $s \in S$, there is a set of decisions, D_s, and a set of random variables, X, the possible values of which correspond to particular events or outcomes. There is exactly one utility variable, v, whose value we wish to make as high as possible by choosing the optimal alternatives for the decisions in $D = \{D_s : s \in S\}$. The expected utility of a scenario is given by $U(s)$. Visually, we denote members of D by square decision nodes, members of X by oval chance nodes, and v by a hexagonal utility node. All these nodes are part of a directed acyclic graph. Arcs whose target is in $X \cup \{v\}$, called *relevance arcs,* represent either probabilistic dependence, or functional dependence of the target variable on the source variable. Arcs whose target is a decision node in D, called *informational arcs,* specify exactly which outcomes and prior decisions are known to the decision maker at the time of the decision corresponding to that decision node. Informational arcs imply a temporal ordering: The information is known before the decision is made. Since we require that there be a total ordering of decisions in each scenario, and that the decision maker does not forget information known at the time of a previous decision, we may abbreviate the diagram by omitting informational arcs from a node to all but the earliest of those decisions that have information about that outcome.

Relevance arcs are labeled with a symbol from $L_E = \{\text{“+”}, \text{“−”}, \text{“0”}, \text{“?”}\}$. For binary (true or false) nodes A and B, a “+” on the arc from A to B indicates that $P(B|A,\xi) \geq P(B|\bar{A},\xi)$. For nonbinary nodes whose random variables have any one of ordinal, interval, or ratio values for their possible outcomes, a “+” indicates that an upward shift in the mean for B's probability distribution, with all other relevant variables held constant, results in an upward shift in the mean for A's distribution. Similarly, a “−” on the arc from A to B indicates that $P(B|A,\xi) \leq P(B|\bar{A},\xi)$ for binary nodes, and that there is an oppositely directed shift in their variable's distribution for nonbinary nodes. A “0” indicates probabilistic independence. Such arcs are generally omitted. A “?” indicates that the direction of such qualitative dependency is nonmonotonic or unclear. Informational arcs should not be labeled in this manner, as the specification of which alternative is optimal should be contained elsewhere in the assessed influence-diagram model.

Contingent nodes play a role only when their explicit conditions on previous outcomes have been met. Following Fung and Shachter's notation, we shall use Ω_j to refer to the joint state space (i.e., all combinations of outcomes) of the variables in the set of nodes $J = \{j_1, j_2, \ldots, j_n\}$:

$$\Omega_j = (\Omega_{j_1} \times \Omega_{j_2} \times \ldots \times \Omega_{j_n}).$$

The direct successors of a node i are those nodes that are targets of arcs originating at node i. The direct predecessors of a node i are those nodes that are sources of arcs terminating at node i. The ancestors of a node i are all nodes that occur prior to i along a path in the directed acyclic graph.

For each node i, there is a joint state space (Ω_K) for all possible outcomes of its ancestors. This state space can be partitioned into component contingency spaces ($\Omega_\kappa = \{\Omega_{1,\kappa}, \Omega_{2,\kappa}, \ldots, \Omega_{n,\kappa}\}$), each of which is defined by a set of conditions. A condition ($\Omega_{i,\kappa}$) for node i is a statement about the outcome of some subset of the ancestors of i. In a contingent influence diagram, more that one node with the same label—representing the same decision or random variable—can be drawn; however, these identically labeled nodes must be associated with mutually exclusive contingency spaces. A contingent node may have one or more contingencies, κ, represented visually by one or more dots on its left side.

References

Barthelmann, K. (1991). Describing Göttler's operational graph grammars with pushouts. In Ehrig, H., Kreowski, H.-J., and Rozenberg, G., editors, *Graph Grammars and Their Application to Computer Science: Proceedings of the Fourth International Workshop*, pages 98–112, Berlin. Springer-Verlag.

Beinlich, I. A., Suermondt, H. J., Chavez, R. M., and Cooper, G. F. (1989). The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In Hunter, J., Cookson, J., and Wyatt, J., editors, *Proceedings of the Sixth Conference on Artificial Intelligence in Medical Care*, pages 247–256. Springer-Verlag, Berlin.

Bradshaw, J. M., Covington, S. P., Russo, P. J., and Boose, J. H. (1990). Knowledge acquisition techniques for intelligent decision systems: Integrating *axotl* and *aquinas* in *dducks*. In Henrion, M., Shachter, R. D., Kanal, L. N., , and Lemmer, J. F., editors, *Uncertainty in Aritificial Intelligence 5*, pages 255–270. Elsevier, New York.

Campbell, K. E. and Musen, M. A. (1992). A system for capturing clinical data based on SNOMED III. Technical Report KSL-92-13, Knowledge Systems Laboratory, Stanford University, Stanford, CA.

Chavez, R. M. and Cooper, G. F. (1990). A randomized approximation for probabilistic inference on bayesian belief networks. *Networks*, 20:661–685.

Clancey, W. J. (1991). Why today's computers don't learn the way people do. In Flach, P. A. and Meersman, R. A., editors, *Future Directions in Artificial Intelligence*, pages 53–62. North-Holland, Amsterdam.

Coté, R. A., editor (1982). *Systematized Nomenclature of Medicine*. College of American Pathologists, Skokiw, IL.

Ehrig, H., Kreowski, H.-J., and Rozenberg, G., editors (1991). *Graph Grammars and Their Application to Computer Science: Proceedings of the Fourth International Workshop*, Berlin. Springer-Verlag.

Ehrig, H., Rozenberg, G., and Rosenfeld, A., editors (1987). *Graph-Grammars and Their Application to Computer Science: Proceedings of the Third International Workshop*, Berlin. Springer-Verlag.

Feder, J. (1971). Plex languages. *Information Science*, 3:225–241.

Finkel, A. J., editor (1990). *Physicians' Current Procedural Terminology*. American Medical Association Press, Chicago, IL, 4 edition.

Göttler, H., Günther, J., and Nieskens, G. (1991). Use of graph grammars to design CAD-systems! In Ehrig, H., Kreowski, H.-J., and Rozenberg, G., editors, *Graph Grammars and Their Application to Computer Science: Proceedings of the Fourth International Workshop*, pages 396–410, Berlin. Springer-Verlag.

Gruber, T. R. (1989). *The Acquisition of Strategic Knowledge*. Academic Press, Boston, MA.

Heckerman, D. E. (1991). *Probabilistic Similarity Networks*. M.I.T. Press, Cambridge, MA.

Herskovits, E. H. and Cooper, G. F. (1990). Kutató: An entropy-driven system for construction of probabilistic expert systems from databases. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, pages 54–62, Cambridge, MA.

Hess, L. and Mayoh, B. H. (1990). Graph grammars for knowledge representation. Technical Report DAIMI PB-304, University of Aarhus, Aarhus, Denmark.

Hollenberg, J. P. (1984). The decision-tree builder: An expert system to simulate medical prognosis and management. *Medical Decision Making*, 4:531.

Holtzman, S. (1988). *Intelligent Decision Systems.* Addison-Wesley, Reading, MA.

Howard, R. A. and Matheson, J. E. (1984). Influence diagrams. In Howard, R. A. and Matheson, J. E., editors, *The Principles and Applications of Decision Analysis*, pages 719–762. Strategic Decisions Group, Menlo Park, CA.

Jimison, H. B. (1990). A representation for gaining insight into clinical decision models. Technical Report STAN-CS-90-1320, Departments of Computer Science and Medicine, Stanford University, Stanford, CA. Departments of Computer Science and Medicine.

Karnopp, D. C., Margolis, D. L., and Rosenberg, R. C. (1990). *System Dynamics: A Unified Approach.* Wiley-Interscience, New York.

Langlotz, C. P. (1989). A decision-theoretic approach to heuristic planning. Technical Report STAN-CS-89-1295, Departments of Computer Science and Medicine, Stanford University, Stanford, CA. Departments of Computer Science and Medicine.

Lauritzen, S. L. and Spiegelhalter, D. J. (1988). Local computations with probabilities on graphical structures and their application to expert systems. *J. Royal Statist. Soc.*, 50:157–224.

Leong, T.-Y. (1991). Representation requirements for supporting decision model formulation. In D'Ambrosio, B. D., Smets, P., and Bonissone, P. P., editors, *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, pages 212–219, Los Angeles. Morgan Kaufmann, San Mateo, CA.

North, D. W. (1968). A tutorial introduction to decision theory. *IEEE Transactions on Systems, Science and Cybernetics*, 4:200–210.

Pavlidis, T. (1972). Linear and context-free graph grammars. *Journal of the ACM*, 19:11–23.

Pearl, J. (1986). Fusion, propagation and structuring in belief networks. *Artificial Intelligence*, 29:241–288.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, San Mateo, CA.

Rousseau, W. F. (1968). Method for computing probabilities in complex situations. Technical Report SEL-68-050, Department of Electrical Engineering, Stanford University, Stanford, CA.

Shachter, R. D. (1990). An ordered examination of influence diagrams. *Networks*, 20:535–563.

Shahar, Y. (1991). Knowledge-acquisition for temporal-abstraction mechanisms. In *Proceedings of the Sixth Workshop on Knowledge Acquisition for Knowledge-Based Systems*, pages 26.1–26.18, Banff, Alberta.

Shwe, M. A., Heckerman, D. E., Henrion, M., Lehmann, H. P., and Cooper, G. F. (1991). Probabilistic diagnosis using a reformulation of the INTERNIST-1/QMR knowledge base. *Methods of Information in Medicine*, 30:241–267.

von Neumann, J. and Morgenstern, O. (1944). *Theory of Games and Economic Behavior.* Princeton University Press, Princeton, NJ.

Wellman, M. P. (1990). *Formulation of Tradeoffs in Planning Under Uncertainty.* Morgan Kaufmann, San Mateo, CA.

Wellman, M. P., Breese, J. S., and Goldman, R. P. (1992). From knowledge bases to decision models. *The Knowledge Engineering Review*, 7(1):35–53.

Wellman, M. P., Eckman, M. H., Fleming, C., Marshall, S. L., Sonnenberg, F. A., and Pauker, S. G. (1989). Automated critiquing of medical decision trees. *Medical Decision Making*, 9:272–284.

Task Modeling with Reusable Problem-Solving Methods

Henrik Eriksson Yuval Shahar Samson W. Tu
Angel R. Puerta Mark A. Musen

Medical Computer Science Group
Knowledge Systems Laboratory
Stanford University School of Medicine
Stanford, CA 94305-5479, U.S.A.

Abstract

Task- and method-oriented approaches to the acquisition and structuring of expertise for knowledge-based systems assume that problems can be decomposed into subproblems that can be solved by clearly understood software modules. We have modeled several standard problems as tasks, and have implemented them using two different problem solvers as an experiment with the use of tasks and problem-solving methods. We have studied the modeling process for the Towers of Hanoi problem and the Sisyphus room-assignment problem by analyzing various approaches that use tasks and methods. This examination revealed that, although both problems can be solved by a problem-solving method implementing chronological backtracking, the goal of casting the method for the tasks required that we pay careful attention to the problem representation, as well as to the adaptation of the method for the problems. There was also a significant difference between the problem-solving methods in the support provided for solving the problems. Our conclusion is that it only makes sense to speak about role-limiting methods relative to the task in question.

1 Introduction

The knowledge-engineering process suffers from a general brittleness, in that the components produced cannot easily be used for other purposes or reused in other similar projects. Many scientific and engineering disciplines have been able to standardize and reuse components such that developers are almost unaware that they are reusing previous work. Traditional engineering disciplines are largely based on systematic reuse of previous work in science and of preceding successful designs. For instance, electrical engineers can choose from a variety of standard components—such as nand gates, adders, and shift registers—in designing circuits. Knowledge engineering does not yet have a set of widely accepted components that can be configured into knowledge-based problem solvers—the kind of reusability found in many traditional engineering disciplines is still a knowledge engineer's dream. Reuse in knowledge engineering can take many forms. Ideally, we would like to reuse expertise from several different areas in many systems. Expertise, however, is by definition domain-specific, so what remains to be reused is knowledge common to several related domains and common-sense knowledge. In addition to reuse of expertise, there also is a demand for reusable problem-solving methods for driving the application system. (We assume that these problem-solving methods operate on a knowledge base that contains the expertise required to accomplish the assignment.)

Establishing the *task* of the application system is an important first step toward finding an appropriate problem-solving method. Before the developer can begin an analysis of potential problem-solving methods for an application system, the developer must understand the task of the system. This *task analysis* leads to a role description for the system, which serves as the basis for the selection of problem-solving *methods* for accomplishing the task. The elicitation, identification, and modeling of appropriate problem-solving methods that can be implemented using computers are generally considered to be three of the most difficult tasks of practical knowledge engineering. It is, therefore, important for developers to be able to reuse previous work in selecting and synthesizing new problem-solving methods. Many researchers

have pointed out similarities among domains and among the methods that can be used to solve problems in these domains (Clancey, 1985; McDermott, 1988; Steels, 1990; Karbach, Linster & Voß, 1990). Such equivalences can be used as a foundation for reusable methods and other components for knowledge-based systems. Several research groups are seeking to develop architectures for reusable problem-solving methods (Steels, 1990; Klinker, Bhola, Dallemagne, Marques & McDermott, 1991; Musen, in press).

The PROTÉGÉ-II project aims to develop automated support for task-oriented knowledge acquisition (Puerta, Tu & Musen, 1992). PROTÉGÉ-II is designed to provide a development environment where tasks can be specified and problem-solving methods can be configured. In addition to supporting development of problem solvers, PROTÉGÉ-II generates knowledge-acquisition tools that elicit the expertise required by the problem-solving mechanisms. Hence, PROTÉGÉ-II supports two important aspects of application-system development: (1) the design of an inference mechanism by method configuration, and (2) the formulation of a knowledge base by generation of an appropriate knowledge-acquisition tool (e.g., a domain-specific knowledge-acquisition tool). The assumption is that the task analysis provides the foundation both for design of inference mechanisms and for generation of knowledge-acquisition tools.

An annoying problem is that there is little consensus on the definition and meaning of *task* and *method,* and that terminological disagreement persists within the knowledge-acquisition community (Karbach *et al.*, 1990). Part of the problem is that related approaches are usually reported from a bird's-eye view—that is, in an abstract form—which makes detailed evaluations and comparisons difficult. In general, it is nearly impossible to derive precise definitions of concepts and to determine the actual behavior of implementations from research papers. The Sisyphus standard problems have certainly provided a common denominator for clarifying different approaches (Linster, 1992). This paper attempts to provide a worm's-eye view of development with tasks and methods by examining one problem in detail.

We use the Towers of Hanoi problem and the Sisyphus room-assignment problem to illustrate task modeling, method selection, and method instantiation. Moreover, we examine how well tasks and methods map onto the problems, and how they support the developer. In particular, we compare several problem-solving methods for the Towers of Hanoi problem, including chronological backtracking and classic recursive solution (recursive-task decomposition). In addition, we present a method for board games that can solve a class of problems in which pieces move between locations, including the Towers of Hanoi problem. To show how methods might be reused, we reuse the chronological-backtracking and board-game methods for the Towers of Hanoi problem in the Sisyphus room-assignment problem. Finally, we discuss several obstacles to use of this approach in the light of the Towers of Hanoi and Sisyphus examples.

This paper is organized as follows: Section 2 provides background of the PROTÉGÉ-II project, and gives a summarized introduction to tasks and methods in PROTÉGÉ-II. Section 3 presents the Towers of Hanoi problem, and examines several solutions to the problem. The use of chronological backtracking as a method for solving the Towers of Hanoi task, is discussed in Section 4. Section 5 presents an approach to reuse the chronological-backtracking and board-game methods for the Sisyphus room-assignment task. The results are presented in Section 7. Section 8 discusses tasks and methods as modeling tools for the problems examined. Section 9 presents the conclusions of the paper.

2 Background

The PROTÉGÉ-II project is not only an approach to tool support for developers of knowledge-based systems, but also an effort to understand how knowledge-based systems can be configured from reusable components and how the knowledge required to solve application tasks can be acquired. The notions of *tasks* and *methods* are central to the design of PROTÉGÉ-II. In the electrical-engineering domain mentioned in Section 1, a task could be to raise an output signal. A circuit designer can accomplish this task by using different types of amplifying circuits. The circuits correspond to reusable problem-solving methods in the PROTÉGÉ-II context.

Generally, tasks are real-world responsibilities that the application program is supposed to discharge. Examples of such tasks follow: (1) given a set of symptoms for a faulty device, produce a diagnosis and a remedy; (2) given an initial state and a goal, produce a plan (i.e., a series of operations) that will accomplish the goal; and (3) maintain at steady values certain measurements and indicators of a manufacturing process over time. Sometimes, a knowledge-based system performing a task will directly replace a human performing the same task; more often, however, the system's task is artificial—a combination of human-

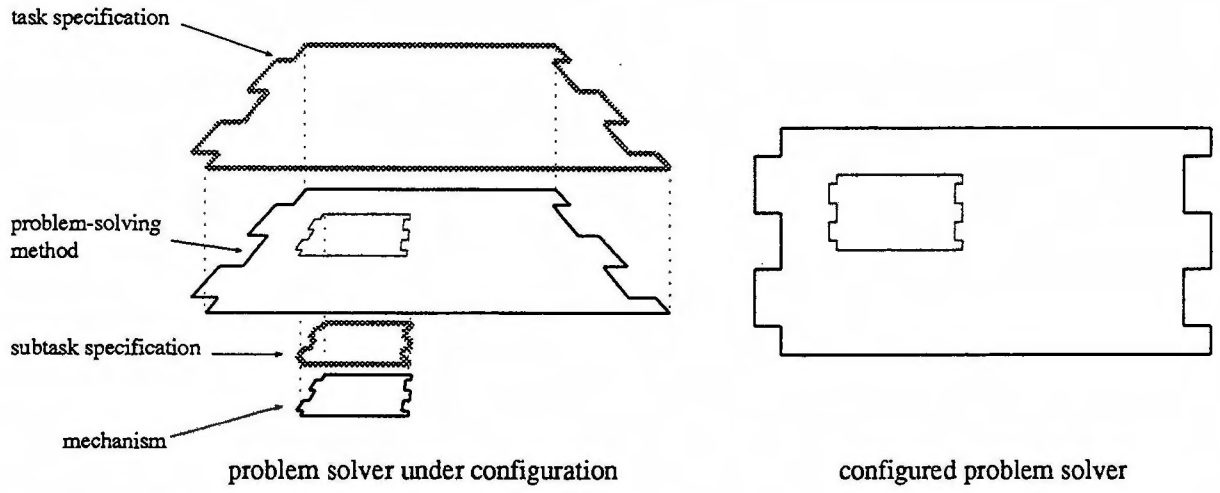


Figure 1: The relationships among task specifications, methods, subtask specifications, and mechanisms in PROTÉGÉ-II (left). The configured problem solver during run time (right).

and computer-oriented subtasks designed to support people in their work. In PROTÉGÉ-II, the features of a task required for identification of appropriate problem-solving methods must be entered by the developer. Currently, this task specification comprises the input and output definition for the task.

Problem-solving methods can be seen as abstract models of how to solve certain problems. In PROTÉGÉ-II, methods are procedures that accomplish tasks. Examples of such problem-solving methods are (1) heuristic classification (e.g., classification of faults given symptoms) (Clancey, 1985), (2) reactive planning (Firby, 1987; Dean, 1988), and (3) propose-and-revise methods for design (Marcus & McDermott, 1989). Often, a specified task can be accomplished by several methods. We can perform troubleshooting tasks, for instance, by matching and classifying faults and by using model-based reasoning. The selection of a method depends on factors beyond the task specification, such as availability of expertise, the time and space requirements for computations, and compatibility with other cooperating methods.

In addition to task specifications and methods, PROTÉGÉ-II features subtask specifications, which are specialized tasks that help methods to complete their tasks, and mechanisms, which are primitive methods that cannot be decomposed further (see Figure 1). The general ideas are that methods solve the problem imposed by the task specification, and that methods may pose new subtasks in the process of accomplishing the overall task. In other words, methods may delegate certain problems as subtasks to be solved by other methods (or by mechanisms). Mechanisms can be viewed as primitive methods that cannot pose subtasks (i.e., mechanisms are capable of accomplishing the task without delegation). From the developer's perspective, in PROTÉGÉ-II, mechanisms are black boxes that cannot be decomposed further.

A distinctive feature of applications for knowledge-based systems is the importance of domain expertise. Problem-solving methods for knowledge-intensive tasks typically require access to domain knowledge to complete their tasks. For example, heuristic classification methods rely on classification knowledge for the domain to be categorized. Such knowledge must be acquired from domain specialists. PROTÉGÉ-II will support the knowledge-acquisition process by providing a meta-tool for generation of domain-specific knowledge-acquisition tools that elicit the expertise needed for the problem-solving methods.

To develop an application system with PROTÉGÉ-II, the developer must (1) analyze the task of the system to be developed, (2) select and configure problem-solving methods that will accomplish the task, (3) analyze subtasks posed by the problem-solving methods, (4) select and configure submethods (or mechanisms) for accomplishing the subtasks, and (5) develop a domain-oriented knowledge-acquisition tool that acquires the expertise necessary for the problem-solving methods to complete their tasks. The steps may be iterated as needed. Note that the result of a task analysis does not specify how a task will be accomplished. Such decisions are made in the subsequent method-selection phase (step 2). Moreover,



Figure 2: The initial state (a) and the goal state (b) of the three-disk Towers of Hanoi problem.

the developer may have to fine tune problem-solving methods by adjusting method parameters to perform their tasks completely. Just as an electrical engineer has to calculate the values of resistors and capacitors in an amplifying circuit, a developer using PROTÉGÉ-II has to adapt the methods selected to the particular application. However, a significant difference between the electrical-engineering and the knowledge-engineering perspectives is that problem-solving methods require expertise to function.

An important question is whether we can identify tasks and select problem-solving methods from a library in the same way that electrical engineers reuse the design of amplifiers for multiple tasks. In the following sections, we shall examine how well-known tasks—most notably the Towers of Hanoi and Sisyphus problems—can be analyzed, and how problem-solving methods can be adapted to accomplish those tasks.

3 Task Analysis and Method Selection: The Towers of Hanoi Problem

In this section, we shall examine the Towers of Hanoi problem, describe the Towers of Hanoi task, and discuss different methods that can be used to solve that task. Our purpose is to provide a simple demonstration of the tradeoffs involved in method selection and the effect of increased insight in the task and the relevant heuristics. The Towers of Hanoi problem is interesting as a case study of a tradeoff of space and time resources versus more task-specific knowledge. It demonstrates several possible task-level indices that can be used by the knowledge engineer looking for a method in the method library, during the task-analysis phase. These indices characterize different dimensions of the solution.

The Towers of Hanoi problem can be regarded as an instance of a more general class of problems—*board-game* problems. Board-game problems include the notions of *locations* and of *pieces* that can be moved between locations according to the rules of the game. We assume that the board game is a one-player game and that there are initial and goal configurations of the game. In the Towers of Hanoi board game, there are k locations, called *pegs,* that can be modeled as stacks. There are n pieces, called *disks.* The move consists of popping one disk from one location's stack and pushing it onto another location's stack. Disks have various sizes, and the local configuration constraint stipulates that disks in every location must form a list ordered by size, the largest disk being on the bottom, the smallest on top. The initial and goal configurations can be arbitrary in general. In what we shall call the *classic version* of the Towers of Hanoi problem (Figure 2), there are three pegs, and the initial configuration is a *tower* configuration—namely, (n, 0, 0). The goal configuration is also a tower configuration—namely, (0, 0, n). In other words, we transfer a tower composed of all the disks from one peg to another, using the third peg, and moving only one disk at a time.

3.1 Dimensions of Methods for the Towers of Hanoi Task

We demonstrate the knowledge and efficiency tradeoffs that are inherent in choosing and configuring a suitable problem-solving method for the Towers of Hanoi task by looking at five different approaches. For each approach, we can ask the following questions:

1. What is the **quality of the solution**? Quality is defined by *internal* and *external* criteria. Internal criteria can be expressed in the domain's terms. For instance, internal criteria might include the

number of moves needed to execute the solution. In general, any ranking function can be used that ranks the quality of the solution with respect to domain-dependent features. External criteria express domain-independent characterizations of the solution, such as conforming more closely to other constraints known to the domain expert, or obeying other domain-independent input and output specifications that are outside the scope of the domain. For instance, external criteria might include the rate of arrival of the input, or the rate of generation of multiple solutions. Another example of a potentially important external criterion is whether solution actions can be executed concurrently.

2. What is the **internal space** needed during execution (computation) of the solution?
3. What is the **internal time** needed to compute the solution?
4. What is the **metaspace** needed for the task agenda, assuming such an agenda is controlling the task-decomposition process and containing the task activations?
5. What is the **metatime** needed to control the task agenda (to decompose tasks, to schedule tasks, etc.)?

3.2 Five Methods for the Towers of Hanoi Task

In the case of the Towers of Hanoi task, at least five problem-solving methods might be considered. We shall briefly discuss the chronological backtracking, recursive-task decomposition, iterative solution, piece-oriented, and general-task decomposition methods.

3.2.1 Chronological-Backtracking Method

The chronological-backtracking method could be used to solve board-game tasks at the PROTÉGÉ-II level. The task specification defines the constraints for the specific Towers of Hanoi task. The chronological-backtracking method uses, in the worst case, a blind, backtracking search for the goal state (assuming backtracking when a previously visited state is reached).

The quality of the solution generated by chronological backtracking, measured by the number of moves, is $\Omega(2^n)$ for the number of moves, where n is the number of disks (the number of moves might be much more than the $O(2^n)$ optimum, if a generate-and-test approach without any heuristics is used). The internal space needed is $\Omega(2^n)$ (for all the configurations stored, needed for recognition of previous configurations). In general, in the worst case, chronological backtracking requires up to $O(k^n)$ space (in units of board size) to store all possible configurations of n distinct pieces and k locations. Of course, the internal time needed to compute the moves, when pure search and comparison with former configurations is involved, is also $\Omega(2^n)$. The agenda metaspace usually will include a constant number of tasks, each task being invoked repeatedly (e.g., generating the next move and checking the constraints). Agenda metatime depends on implementation: It could be constant, but it might be up to $O(k^n)$ if every dead end forces a new invocation of the chronological-backtracking subtasks.

Note that chronological backtracking does not supply us with an immediately executable recipe, since it includes, by definition, backtracking. Chronological backtracking is a trial-and-error method. Furthermore, we can say little about what should happen, for instance, in the ith move to the jth piece. Chronological backtracking is, however, a general method, and depends on little task knowledge, except for basic domain-dependent definitions that map the board-game ontology to the domain at hand.

3.2.2 Recursive-task Decomposition

In general, to specify a solution by recursive-task decomposition (RTD), we need a base case and a specification of the recursion in terms of the input task and of that task's decomposition into simpler tasks of the same nature. For instance, we could decompose the classic Towers of Hanoi task in the following way, to move a tower of n disks from peg A to peg B using peg C:

```
classic_transfer(n, A, B, C) =
   {if (n = 0) then return else
```

```
    {transfer(n-1, A, C, B);
     move(1, A, B);
     transfer(n-1, C, B, A)}
}
```

where `move(1, A, B)` is the basic move task defined for the Towers of Hanoi board game. RTD would need, in this case, $O(n)$ metaspace for all the intermediate tasks, and $O(2^n)$ metatime for the task agenda, assuming that each task needs a fixed space, including the basic move task; $O(2^n)$ internal time for computing the moves, assuming constant time per basic move; and no internal space for remembering states. The quality of solution, measured by number of moves, is $O(2^n)$ (optimal).

Note that RTD requires considerable domain and task-specific knowledge to suggest a viable decomposition leading to a legal and high-quality solution. RTD assumes, for instance, knowledge about the correctness of the algorithm implicit in the decomposition of the classic Towers of Hanoi version. RTD outlines implicitly the whole solution. However, we cannot immediately answer questions such as "what should I do now?" since we have to unfold the whole plan to execute even the first move task.

Note also that the specific RTD solution outlined could work for any tower configuration of the Towers of Hanoi problem and for any value of n or k (without using the additional pegs); for only the classic version, however, does it produce the optimal solution. This approach would not work for a general configuration.

3.2.3 Iterative Solution

In an iterative method, the user has to supply a set of rules, or an algorithm, that provides (in the case of a board-game task) a direct transformation from one configuration to another, starting with the initial configuration and ending with the goal configuration. One such instantiation of the iterative method is the following simple set of rules, where the main idea, incorporating a deeper analysis of the specific classic version, is derived from a *topological* representation of the board.

Represent the pegs as a cycle (in the classic case, the triangle A, B, C). The smallest disk moves alternately around the cycle, clockwise if n is odd and counterclockwise if n is even. In between its moves, the only other legal move (by the current second-largest movable disk) is made.

The iterative solution needs constant metaspace and metatime to manage a single task. The solution quality, in terms of number of moves, is the optimal number of moves, $O(2^n)$. Constant internal space, using the size of the board as a unit (for the current configuration), is needed. Internal computation time is $O(2^n)$.

Note that the iterative method supplies an external quality to the solution: It is executable, in the sense that we do not, strictly speaking, produce a plan (unless we save a list of moves); rather, we obtain an execution trace of the optimal solution. Given the current configuration and the last move, we know exactly what to do immediately (i.e., we know the answer to "what should I do now?"). We cannot, however, ask directly "what should disk i do at step j?" without running the whole process to answer the question. The iterative solution in this form is specific to Towers of Hanoi configurations of the tower type, and is optimal for only the classic Towers of Hanoi task.

3.2.4 Piece-Oriented Method

The piece-oriented method, useful for a small class of board-game tasks, is not a configuration- or location-oriented method. Rather, it can be seen as an object-oriented version of an iterative algorithm that also has a useful parallel-execution quality. The user must supply a uniform set of rules that directly maps the internal state of the piece (including *time*) into a possible move. The total configuration either is not used, or is used in a limited fashion. The following set of rules is adapted from a temporal analysis of the Towers of Hanoi problem (Hale, 1987), and uses both a topological and a temporal representation:

1. Number the disks $(0, 1, \ldots, n-1)$, and number the (discrete) time units $(0, 1, \ldots, 2^{n-1})$.
2. $\forall i$, Disk i starts moving at time unit 2^i.
3. $\forall i$, Disk i moves, every 2^{i+1} time units after its first move.

The pegs are represented as a cycle. Each disk moves one location to the left or to the right each time in the same direction, depending on the parity of $(n - i)$: clockwise if it is odd, and counterclockwise if it is even.

No internal space is needed to preserve the current configuration, since moves are of low complexity and are outlined in advance for each piece. The solution quality is $O(2^n)$, if measured only by the number of the moves (optimal solution). The internal time is $O(2^n)$. Metaspace and metatime are constant (one task).

Note that *concurrent* moves are computed, starting from the initial state. A disk moves when a global (or internal) counter sends it the appropriate temporal message (i.e., the two mentioned criteria are fulfilled). This approach adds another (external) dimension to the quality of the solution—concurrent processing of the moves. Since we know ahead when and to where each disk moves, we can, in theory, exploit the possibility of *parallel execution,* where several disks can be moved independently from and to different locations (other semantics of concurrent executions are possible). Concurrent execution is important if, for example, we execute the Towers of Hanoi task using a two-handed robot that can move several disks at a time. The independent, object-oriented, parallel set of plans implicit in such a method is especially important if the pieces represent several robots that have to move independently, without the robots knowing the total configuration at any stage.

Note that, in this piece-oriented solution, we can ask questions such as "when does disk i move for the first time?" or "when, and in what direction, could disk i move on step j?" or "which are the disks eligible to move on step j?" We could not ask such questions in the previous solutions. Thus, a temporal dimension has been added to the quality of the solution. The concurrent-processing solution, in this form, is optimal only when applied to the classic Towers of Hanoi task; it can be used for any k, but, unless generalized, it will not be optimal.

3.2.5 General-Task Decomposition

General-task decomposition (GTD) is a more general version of RTD. In GTD, we try to decompose the task explicitly into several subtasks. One or more of these subtasks could include a simplified version of the original task, but they might simply be tasks that can be solved by mechanisms existing in the method library or that can be further decomposed into solvable subtasks.

Using GTD, we can solve the general task of transferring any initial configuration C_0 to any goal configuration C_g, by decomposing the Towers of Hanoi task into the following subtasks, both utilizing a new task, `make_tower(C)`, that transfers a configuration into a single tower, and by using a general operator, *reverse,* that reverses a plan by applying the inverse of all move operators in reverse order:

```
transfer(C0 , Cg) =
        {make_tower(C0);
         reverse(make_tower(Cg))}.
```

The `make_tower(C)` task, however, is decomposed easily into n tasks that solve the classic version for m disks, $m = 0, 1, \ldots, n - 1$, since transforming any configuration C into a single tower involves creating a tower of one (smallest) disk, then transferring it (if necessary) to the top of the second smallest disk (which must be free), then transferring the tower of two (smallest) disks (if necessary) on top of the third smallest disk, and thus eventually transferring a tower of $n - 1$ disks onto the largest disk. Hence, we can use any of the specific methods used for the classic version of the Towers of Hanoi problem to solve the more general Towers of Hanoi task. Our solution depends on only the method used to solve the classic version.

Since each classic transfer subtask uses $O(2^k)$ time, $k = 0, 1, \ldots, n - 1$, the total internal time needed for solving the `make_tower` task is $O(2^n)$, which is also the internal time needed for computing the moves for the general transfer task, assuming constant time per move. The quality of the solution, measured by the number of moves, is not necessarily optimal, but it is still $O(2^n)$. Constant internal space, using the size of the board as a unit, is needed for saving the most recent configuration, and for instantiating the pegs correctly. Metaspace is constant, assuming constant space per task, and assuming an iterative formulation of `make_tower`, where no more than one classic task is being handled (decomposed) at any time by the agenda, and assuming an iterative method is chosen. Metatime is $O(n)$, if method 3 or 4 is used.

Note that the decomposition described here resembles the use of a macro-operator problem solver (Korf, 1985), whose single operator transfers any state into one intermediate state. Unlike the specific RTD solution presented in Section 3.2.2, the more general GTD method works for any initial or goal configuration in the Towers of Hanoi task. Indeed, it works for any board-game task, if a suitable intermediate configuration and appropriate operator can be found (such as a single tower in the Towers of Hanoi task, amounting to transferring all pieces to one location and rearranging them back into the goal configuration). Another prerequisite for applying this method is that the basic move operator of the board game has an inverse; thus, the reverse operator can work. In the case of the Towers of Hanoi task, this specific decomposition will also work for any k and n, although it might not be as efficient in the number of moves as is the optimal solution (or even a solution found through chronological backtracking); even in a three-peg Towers of Hanoi task with general initial and goal configurations, it might use up to twice the optimal number of moves—including the classic-version case. This phenomenon is common in solutions using macro-operators, where the worst-case solution length is no more than the optimal solution times the number of subgoals to which the problem is broken down (Korf, 1985). As in RTD, finding a good decomposition is a crucial step and might require that we acquire decomposition knowledge from a domain expert (or possibly that we acquire it through learning, as is possible in certain domains (Korf, 1985)).

Note also that we cannot necessarily ask in this method "what should I do now?" since the task is solved by a series of subtasks, some of which might be recursive; at least, however, we have a skeletal plan, and we know the answer to "what subtask should I achieve now?" We can also instantiate this skeletal plan in several ways, to get a recursive, iterative, or concurrent computer solution. In this sense, the GTD solution is the most flexible.

4 Method Configuration of Chronological Backtracking

To test our ideas on the PROTÉGÉ-II problem-solving architecture, we encode the classical search algorithm of *chronological backtracking* as a mechanism in the PROTÉGÉ-II method library. As a weak (general) method, chronological backtracking makes little commitment on the problem representation of the problem data. In particular, the method makes no assumption about the representation of a state. The only requirement the method places on the definition of a state is that an equality predicate can be defined for any two states. To detect and avoid circularities in the space of states, the algorithm for chronological backtracking uses the equality predicate to check whether the current state on its search path is equal to a state that is already on the path. Thus, to use chronological backtracking to solve different problems, the developer must define the concept of the state and the equality predicate from scratch each time.

4.1 Chronological Backtracking as a Reusable Method

Key to the success of the chronological backtracking algorithm is the ability to generate the legal next states from any given state. In our first attempt to formulate the chronological backtracking as a mechanism, we defined, as an input to the mechanism, the function that enumerates the possible next state from a given state. We called the function a *transition function (T-function)*. However, our inability to represent such a function as a declarative input data structure led us to reformulate the problem of finding the next states from a given state as a subtask of the chronological-backtracking *method*. Similarly, in our initial statement of the method, we considered, as an input to the method, an optional *sorting function (S-function)* that reorders the output of the T-function. This sorting function encodes domain heuristics that reorder the set of possible next states so that states that are more likely to be on the solution path are tried first. Thus, the use of an S-function improves the efficiency of the search for a path to the goal state. Currently, we formulate the problem of sorting the output of a T-function as a separate task. Our model of the chronological backtracking method has the following components:

1. The ontology of the method consists of a description of states in the problem to be solved and an equality predicate defined between any two states.

2. There are three *inputs:* (1) an initial state S_I, (2) a goal state S_G, and (3) constraints on legal states. We assume that the constraints on legal states will be defined at the method-configuration time. However, prior to the definition of specific representation for the states, we can make no assumption

about the point in the knowledge-acquisition process at which the initial and goal states will be defined.

3. The *output* of the method is a list of states $\{S_1, \ldots, S_k\}$ that satisfies the following conditions:
   (a) $S_1 = S_I$.
   (b) $S_k = S_G$.
   (c) For each i such that $1 \leq i \leq k$, S_i satisfies the legal-state constraints $C(S_i)$; and, for each i, $1 \leq i < k-1$, S_{i+1} is a member of $T(S_i)$, where $T(S_i)$ is the output of the next-states subtask, given input S_i.

4. The method has two *subtasks:* the *next-states* subtask $T(S_i)$ and an optional *sort-states* subtask. The next-state subtask requires, as inputs, a state S_i, and a predicate $P(S_i, S_j)$. The output of the task is a set of states S_j that satisfies the predicate $P(S_i, S_j)$. The input of the sort-states subtask is a list of states, and the output is a reordered list of the states on the input list. Both the next-states subtask and the sort-states subtask have the same ontology as that of the chronological-backtracking method.

5. The algorithm of the chronological-backtracking method can be expressed as follows:
   (a) Define two internal variables X and Y such that X keeps track of all states that the algorithm has considered already and Y is a list of possible next states. Start the algorithm by setting Y to the list consisting of the initial state S_I.
   (b) If Y is the empty list, then there is no solution and stop.
   (c) If the first state of Y is the goal state S_G, then add the state at the end of the X variable, return the value of the X as the solution, and stop.
   (d) If the first state of Y is a member of X, then delete the first state from Y, and go to step 5b.
   (e) If the first state of Y is not a member of X, then post the subtask *next-states* with the first state of Y as input, and go to step 5b.

4.2 Configuration of the Chronological-Backtracking Method for the Towers of Hanoi Problem

To apply the chronological-backtracking method to the Towers of Hanoi problem, we must formulate the problem as a task. To make the problem less trivial, we have generalized the Towers of Hanoi task to that of achieving a specified configuration of disks on pegs from an initial configuration. This formulation is general enough to be nontrivial, yet specific enough for us to come up with a detailed solution.

To model the Towers of Hanoi problem, the developer, at PROTÉGÉ-II level, defines the disk-moving task and specializes chronological backtracking as the solution method for the task. The chronological-backtracking method requires that we define the concept of *state* and an *equality predicate* for states. We represent the pegs as a set of state variables, $\{v_1, v_2, \ldots, v_n\}$, each of which can take as its value a list of positive integers or the empty list. The integers represent the disks on the peg, and the integer values represent the sizes of the disks. The empty list indicates that no disk is placed on the peg. A state in the disk-moving task is represented by the values of all of the state variables. Two states are equal if the values of all state variables in one state are equal to the values of the corresponding state variables.

The chronological-backtracking method requires that we define a mechanism for the next-states subtask (see Section 4.1). In the disk-moving task, we define the next possible states as the result of moving one disk from one peg to another. We define the domain operation move(v_i, v_j) that moves the top disk of peg i to the top of peg j. The move operation is defined in terms of primitive operations on the list datatype in the ontology of the method. The operator can be defined as

```
push(vi, vj);
pop(vj)
```

We can define a mechanism for the next-states subtask T(S) as the union of the results of applying all possible moves from each state variable. That is, for each state variable v_i, the set of i and j permutations, where $i \neq j$, represents all the possible actions of moving one disk from peg i to peg j. Thus, for any state S, if we define $S_{ij}(S)$ to be the result of applying move(v_i, v_j) to S, then we can define the T-function to be the function

$$T(S) = \{S_{ij}(S) \mid S_{ij}(S) \neq S \wedge P(S, S_{ij}(S)) \wedge C(S_{ij}(S))\}$$
, where $i = 1, \ldots, n; j = 1, \ldots, n; i \neq j$

To complete our modeling of the problem-solving method for disk-moving task, we need to define (1) the number of pegs n, (2) the initial state S_I and goal state S_G (both states are represented as particular assignments of values to the n state variables), and (3) the predicate C(S) that represents constraints on permissible states. In this model, our example Towers of Hanoi problem (Figure 2) has the following definition:

1. $n = 3$.
2. $S_I = \{v_1 = (1, 2, \ldots, m-1, m), v_2 = (), v_3 = ()\}$,
   $S_G = \{v_1 = (), v_2 = (), v_3 = (1, 2, \ldots, m-1, m)\}$, where m = number of disks.
3. The states must satisfy the state-consistency constraint C(S) that says that, for each state variable v_i in a state S, if the value of v_i is $(d_{1,i}, d_{2,i}, \ldots, d_{k,i})$, then $d_{j,i} > d_{j+1,i}$.

Note that, with this definition of C(S), we do not have to constrain the moves of the game with P(S_i, S).

5 Chronological Backtracking Reused: Sisyphus

To demonstrate the reuse of the chronological backtracking method, we implemented the Sisyphus room-assignment problem in the framework that we used for the Towers of Hanoi problem. Furthermore, we developed for the problem a prototype knowledge-acquisition tool, which works together with the problem-solving method.

5.1 Task Analysis

Let us first briefly recapitulate the Sisyphus room-assignment problem. A research group is moved to a new location, and offices must be assigned to persons. The role of the expert system is to allot rooms to persons in the group given information about the staff, descriptions of the offices, and a set of constraints. Individuals have their own professional characteristics and personal preferences (e.g., role, current project, or use of tobacco).

We assume that the information available at run time consists of a set of person descriptions and a set of room descriptions. Tables 1 and 2 show attributes for persons and rooms derived from the Sisyphus problem definition. (If the system will be used several times, it does not make sense to regard such data as part of the knowledge base.) In the Sisyphus problem description, knowledge is expressed as a transcript of a problem-solving session where a human expert assigns persons to rooms. Each assignment step is explained by a brief comment in natural language—for instance, "the heads of large projects should be close to the head of group and the secretariat." Clearly, this transcript describes an idealized knowledge-acquisition situation, with most of the distracting factors of real-world knowledge acquisition removed.

Even for a relatively simple problem, such as the room-assignment problem, many design decisions must be made in the task-analysis phase. The role of an expert-system application in this domain is to replace the current expert in room assignment. (Alternative roles could be to simulate consequences of various assignments, to train novices in room assignment, and to critique solutions.) The input and output to the expert system must be determined. In the Sisyphus problem, we regard the descriptions of the staff and of the rooms as input to the system. The output of the system is a mapping from persons to rooms (i.e., a set of room assignments). Another important factor for the task analysis is the availability of expertise. In this case, the knowledge is available in the form of transcripts from problem-solving sessions. Assignment constraints have to be abstracted from the transcript, and these constraints will influence the

Table 1: Attributes of persons in the Sisyphus room-assignment problem.

attribute	description
`name`	what is the name of the person
`role`	what is the professional role of the person (e.g., researcher or secretary)
`smoker`	whether the person smokes tobacco
`hacker`	whether the person is doing programming
`works with`	who the coworkers are
`single office`	whether the person is eligible for a single office

Table 2: Attributes of offices in the Sisyphus room-assignment problem.

attribute	description
`name`	what the name or number of the room is
`size`	what the size of the office (small or large) is
`central`	to what degree the room is located centrally in the suite

modeling of a problem-solving method for this problem. For instance, we cannot use a machine-learning approach easily, since there is only one sample solution.

Clearly, the Sisyphus room-assignment task can be modeled and implemented in many ways. The variety of contributions to the European and Banff knowledge-acquisition workshops shows that it is possible to cast this problem for many frameworks. In many ways, the Sisyphus problem is similar to the Towers of Hanoi problem. For instance, neither problem requires great expertise or numerous heuristics. Also, much of the task analysis is presupposed in both problems, because it is part of the problem definitions. Brute-force approaches that enumerate all the possible solutions and choose the best solution from that list are feasible for both problems. Hence, there is no significant knowledge-elicitation problem for these artificial problems. There is, however, a modeling problem in terms of developing the problem ontology and suitable operations on this ontology that will solve the problem.

5.2 Chronological Backtracking for the Room-Assignment Task

Having analyzed the Sisyphus task, we continue with selection and instantiation of a problem-solving method. The output of the task analysis is a description of what the system should do, as well as an input and output specification for a problem solver. In the method-selection phase, the developer selects—or, if required, implements—a problem-solving method that accomplishes the task. In this exercise, however, we presuppose chronological backtracking as the problem-solving method.

The Sisyphus room-assignment problem can be solved by the chronological backtracking method, if the stages in the assignment process are seen as states. Basically, we can see the problem situation as beginning with a number of persons standing outside a building, the task being to bring persons to their rooms one at the time such that all constraints are satisfied. In this view, states describe persons outside the building and persons already assigned rooms. Each move in this game formulation is to bring a person inside the building. The output is a plan that describes how to move people into the building. The output of the Sisyphus task, however, should be a set of assignment of persons to rooms—that is, the final state of the plan.

Several data structures must be defined if the problem-solving method is to work. In addition to the person and room representations (Tables 1 and 2), a representation for *states* in the room-assignment domain is required by the problem-solving method. Figure 3 shows a definition of a Sisyphus state. The class `sisyphus-state` inherits from the `state` class. The flag `goalp` indicates whether the goal of assigning persons to rooms correctly has been achieved. The slot `assign-person` holds the person to be assigned an office in this state. The slot `considered-rooms` contains a list of rooms considered for the person to be assigned a room (i.e., `assign-person`). The slot `assign-room` holds the room selected eventually

```
(defclass sisyphus-state (is-a state)
          (slot goalp (default FALSE))
          (slot assign-person)
          (slot considered-rooms (multiple))
          (slot assign-room)
;         (slot assignments (multiple))
          (slot unassigned-persons (multiple))
          (slot available-rooms (multiple))
          (slot available-shared-rooms (multiple))
          (slot smoker-rooms (multiple)))
```

Figure 3: A specialization of the `state` class for the Sisyphus room-assignment problem. (The `sisyphus-state` is defined using CLIPS (NASA, 1991).)

for the person in question. The slot `unassigned-persons` contains a list of persons not yet assigned a room. The slots `available-rooms` and `available-shared-rooms` maintain lists of free rooms and shared rooms where there is space for another person (room-mate) respectively. Finally, the slot `smoker-rooms` holds a list of all (assigned) rooms occupied by smokers. As might be obvious from this description of Sisyphus states, we store only one person–room assignment pair at each state. Initially, we intended to store the complete set of assignment pairs at each state, but this approach resulted in redundant data and additional code for maintaining lists of assignments. Hence, we commented out the `assignments` slot in the state definition. (Note that the program can compute the complete set of assignments for a state by taking the union of assignments in all the preceding states.)

The T-function for the room-assignment problem is more complicated than is that for the Towers of Hanoi problem. The general idea behind the T-function in the former problem is to pick a person from the list of unassigned persons and to find a room suitable for that person. To select a room for a person, the T-function maintains a list of rooms considered for the person (`considered-rooms`). Initially, the list holds the union of all available and shared (but not filled) rooms in the current state, or, if the person is eligible for a single room, of all available (empty) rooms only. As more constraints are enforced, inappropriate rooms are removed from the `considered-rooms` list (e.g., if the person is a nonsmoker, all the shared rooms occupied by smokers are removed from the `considered-rooms` list). More specifically, the T-function

1. Selects the first unassigned person
2. Makes a list of rooms to consider for this person using constraints acquired from the domain expert
3. Selects a room from the list of rooms considered
4. Instantiates a new `sisyphus-state` and copies the relevant information from the current state to the new state
5. Removes the person and room to be assigned from the lists of unassigned persons and available rooms for the new state
6. Adds the room to the `smoker-rooms` list if the person is a smoker
7. Returns to step 3 until all rooms have been considered

Since there is no explicit final state in the room-assignment problem, we cannot determine whether the problem-solving methods has reached its goal by simply comparing the current state against the goal state. The goal state is the solution for this kind of problem, and the goal is stated *implicitly* by a set of criteria that the goal state should fulfill. In this case, we can determine whether the algorithm has reached its goal by simply checking whether all personnel have been assigned to an office (e.g., by testing whether the slot `unassigned-persons` is empty in the state in question). This approach assumes that the T-function

always returns legal person–room assignments for every state. If a legal next state cannot be found, the algorithm must backtrack, and revert to a previous state.

Normally, the chronological-backtracking method must check for circularities in the states generated to avoid infinite loops. In the room-assignment case, however, we do not need to check for any circularities, because persons are taken from the **unassigned-persons** list as they are assigned to rooms. Thus, the algorithm cannot continue when the list is empty, and no circularity check is necessary. Also, since there is no circularity check, there is no need to define an **equal** function for comparing whether two states are similar.

5.3 Support for Knowledge Acquisition

Analogous to the task analysis, the development of a domain-oriented knowledge-acquisition tool must be preceded by a *knowledge-acquisition analysis*. The purpose of the phase of knowledge-acquisition analysis is to examine the development situation from a knowledge-acquisition point of view and to formulate requirements for tool support. This analysis involves (1) the identification of domain experts (e.g., tool users), (2) the segmentation of the part of the knowledge base that is to be acquired through the tool, (3) the definition of a language in which to express the knowledge (e.g., a graphical language), and (4) the specification of semantics for this language, as well as, of denotational semantics that describe the generation of knowledge bases. In other words, knowledge-acquisition analysis is a phase of design of support tools, or alternatively is a selection of a preexisting tool.

When the knowledge-acquisition situation has been analyzed and the role for the knowledge-acquisition tool has been established, the developer can implement the tool. If we examine the Sisyphus room-assignment problem and the sample transcript provided, we find that the expert is concerned not so much with the individuals themselves, as with their *roles* in the group. The transcript contains statements such as: "The head of the group needs a central office." (The fact that a particular person is the director of the group must then be described in the run-time input data.) Another important observation is that the roles provide a specification, or sometimes a justification, for the type of room to which a person should be assigned—for instance, the director of the group requires a large single room, whereas the staff researcher may share a room with another person. Hence, our hypothesis is that much of the domain-knowledge required for room assignment can be expressed in the form of rules where the premise matches a certain role, and where the rule conclusion is a room specification (e.g., a query to a database of rooms available). We are primarily interested in expressing a mapping from roles to potential assignments of persons to rooms. An example of a rule in this rule set follows:

$$\forall(p, r)\ (p \in \text{person} \wedge r \in \text{room})\text{: head-of-group}(p) \rightarrow$$
$$\text{potential-assignment}(p, r) \wedge \text{large}(r) \wedge \text{central}(r)$$

Because the chronological backtracker uses the notion of states for describing the current assignments, we must add state as context information to the rules—that is,

$$\forall(s, p, r)\ (s \in \text{state} \wedge p \in \text{person} \wedge r \in \text{room})\text{: head-of-group}(p) \rightarrow$$
$$\text{potential-assignment}(s, p, r) \wedge \text{large}(r) \wedge \text{central}(r)$$

Since CLIPS (NASA, 1991) is used as the implementation language, these rules must be expressed in the CLIPS format. The following is the actual CLIPS rule that we use for asserting a query for possible rooms in the CLIPS fact base given a person role:[1]

```
(defrule siggi-1
  (query ?s ?p head-of-group) =>
  (assert
    (room-query (state ?s) (size large) (sharing FALSE) (central TRUE))))
```

The goal of the knowledge-acquisition tool is to acquire this type of rule from the domain expert. Interview-oriented knowledge-acquisition tools—such as ETS (Boose, 1985), MOLE (Eshelman, Ehret, McDermott &

[1] In addition to the *central* and *large* requirements on the room, it is necessary to specify whether or not the person should share a room.

Table 3: A sample table of person roles and the room properties requested for them.

person role	room size	sharing	central room
head-of-group	large	false	true
secretary	large	true	true
researcher	large	true	—

Tan, 1987) and MORE (Kahn, Nowlan & McDermott, 1985)—would have asked questions about various roles in a textual dialogue, such as, "Should the director of the group have a small or a large room?" In this case, however, we prefer an explicit language for this type of knowledge, because implicit representations (e.g., a set of such questions posed to the expert) do not give a good overview of the knowledge.

Assuming that the expert cannot program in CLIPS, we must find a language in which the expert can express the relevant knowledge. This language can, for instance, be a restricted form of CLIPS rules. Other alternatives include macro expansion from a simple rule format into CLIPS rules (however, the current version of CLIPS does not support macros) and compilation from a domain-specific language (e.g., restricted natural language) into CLIPS rules.

A closer examination of the rules reveals that the rule set can be expressed in the form of a table, where the roles and the room properties desired can be expressed on one row (see Table 3). Note that arbitrary rule sets cannot be expressed as tables—to be expressed in this type of table the rules must follow a certain pattern. On the other hand, arbitrary rule sets are, in a way, a general programming language because they are Turing complete—and, since the experts are assumed to be nonprogrammers, we cannot require them to write arbitrary rule sets.

What we are doing in this knowledge-acquisition analysis is creating a domain-specific, and possibly visual, language for experts. Designing such a high-level language should be a cooperative process undertaken by the developers and the expert. The developers can then create knowledge-acquisition tools that implement the language and that transform the language into knowledge bases.

At this point in the development process, it is possible to create an overall design of the emerging knowledge-acquisition tool. The analysis of the knowledge-acquisition situation reveals that most of the declarative knowledge needed could be separated and expressed in a table. We chose to build the knowledge-acquisition tool around an editor for tables of role and room properties similar to Table 3. Typically, a knowledge-acquisition tool based on graphical knowledge editing contains several knowledge editors that support acquisition of the structures required for the knowledge base. In this simple case, however, we need to provide only an editor for tables. Only one instance of the table editor is necessary, since the Sisyphus room-assignment problem can be fitted into one table.

The next stage is to determine the dialog structure for the knowledge-acquisition tool—a task that generally requires careful attention, but is simple in this case. The user must be able to access the table editor, for instance, from the main menu of the tool. In addition to providing access to the table editor, the main menu must supply services such as persistent storage of tables (e.g., by saving and loading tables on files) and generation of target knowledge bases—in this case, CLIPS rules.

When the dialog structure is defined, the developers can proceed with the detailed design and implementation of the knowledge-acquisition tool. To illustrate how a knowledge-acquisition tool custom-tailored for the Sisyphus room-assignment problem can be designed according to this analysis, we have implemented a prototype tool in the meta-tool DOTS (Eriksson, 1991). DOTS supports specification of knowledge-acquisition tools according to an architectural model of the target tool. The table of person roles and room properties (Table 3) has been implemented as a form-based knowledge editor in DOTS, and the conversion from table information to CLIPS rules has been implemented as a set of transformation rules. Figure 4 shows the resulting table editor and the main menu for the Sisyphus knowledge-acquisition tool.

Much of the implicit information in the sample transcript in the Sisyphus problem definition can be asserted in the table editor. The first column in the table (the left window in Figure 4) allows the user to specify the professional roles (e.g., head-of-group, secretary, and manager). The second column of the table can be used to specify room properties for smokers and nonsmokers. (Also, the user can

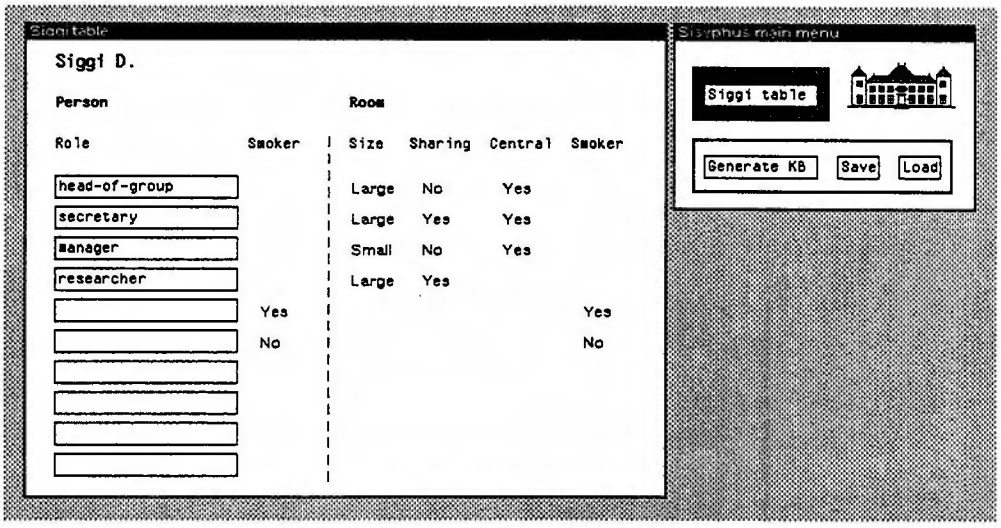


Figure 4: A screen display from the knowledge-acquisition tool for the Sisyphus room-assignment problem. The left window is the table editor for roles and room properties; the right window is the main menu for the tool.

express conjunctions of the roles and the smoking properties. For example, it is possible to specify room requirements for smoking researchers.) The room-specification part of the table features the size, sharing, centrality, and smoker properties. The positions in the size column can be denoted as small, large, and blank. A blank position in the table expresses an irrelevant property value (i.e., a wild card). The sharing, central, and smoker properties are Boolean, and the corresponding column positions accept the values yes, no, and blank. The main menu (the right window in Figure 4) contains four buttons. The "Siggi table" button opens the table editor, and the "Generate KB" button opens a dialog window for the knowledge-base generator. In addition, the buttons "Save" and "Load" handle file input and output for the table contents.

A sample output from the knowledge-acquisition tool is shown in Figure 5. Note that each line in the table in Figure 4 corresponds to a rule in the generated rule set (rules `siggi-1` through `siggi-6`). Thus, the number of rules is equal to the number of rows in the table. The user enters new professional roles in the knowledge-acquisition tool by adding new rows in the table. The number of columns, however, is fixed, and new person and room properties cannot be added by the expert.

In summary, this knowledge-acquisition tool provides support for entering and editing of Sisyphus expertise in the form of tables describing room requirements for professional roles. The generality of this knowledge-acquisition tool is limited. The tool allows its users to create new professional roles and to specify room requirements for the roles. However, the tool does not support addition of new person and room properties. Currently, such changes have an effect on both the knowledge-acquisition tool and the T-function of chronological backtracking. Also, there are several areas of the Sisyphus problem that are not covered at all by the knowledge-acquisition tool. For instance, the user cannot describe different goals of the assignment, or specify in what order persons are assigned to rooms. Currently, such decisions are hard coded in the T-function. Fundamental changes to the problem definition—such as taking into account personal preferences concerning the north and the south side of the building when assigning rooms—can be handled by neither the knowledge-acquisition tool nor the problem-solver configuration (i.e., the T-function).

The knowledge-acquisition tool is coupled to the problem-solving method—that is, the output of the tool is merged with the problem solver's knowledge base. If the problem solver is changed significantly, the knowledge-acquisition tool may have to be modified as well. This interchange can be both a strength and a weakness. It confers an advantage since it may allow the knowledge entered in the tool to be reused (provided that the tool is fashioned to the new problem solver), and it represents a disadvantage if the

```
(defrule siggi-1 (query ?s ?p head-of-group) =>
(assert (room-query (state ?s) (size large) (sharing FALSE) (central TRUE))))

(defrule siggi-2 (query ?s ?p secretary) =>
(assert (room-query (state ?s) (size large) (sharing TRUE) (central TRUE))))

(defrule siggi-3 (query ?s ?p manager) =>
(assert (room-query (state ?s) (size small) (sharing FALSE) (central TRUE))))

(defrule siggi-4 (query ?s ?p researcher) =>
(assert (room-query (state ?s) (size large) (sharing TRUE))))

(defrule siggi-5 (query ?s ?p ?) (test (slot-value ?p smoker)) =>
(assert (room-query (state ?s) (smoker TRUE))))

(defrule siggi-6 (query ?s ?p ?) (test (not (slot-value ?p smoker))) =>
(assert (room-query (state ?s) (smoker FALSE))))
```

Figure 5: Sample output from the knowledge-acquisition tool for the Sisyphus room-assignment problem.

knowledge-acquisition tool cannot be shaped for the new problem solver.

6 A Refined Problem-Solving Method: The Board-Game Method

The idea of the board-game method is to view the problem as a board game where *pieces* can be moved between *locations*. We assume that the game has a fixed number of pieces and locations. More than one piece can be moved to the same location simultaneously, and, if required, the pieces at each location can be ordered. The notion of states is not as important in the specification of board games as it is in the definition of the T-function in chronological backtracking. The states of the game play only a minor role, except for the initial and goal states, which are needed to express the start and finish of the game. Since many games have a large number of goal states, the goal can be described optionally as a goal predicate that checks whether a certain state is the goal. For example, chess has a huge number of goal states, but there exists a simple rule for determining whether the game has been won. Furthermore, there is a notion of *moves*, where pieces are moved between locations. Several pieces may be transferred in each move—that is, a move can consist of a number of *actions*. Legal moves are defined by constraints on how pieces move between locations. Here, there is no explicit notion of a transition function that transforms a state into the next state. Rather, actions represent the withdrawal of a piece from the source location and the deposit of the piece at that target location.

Moves (including all their actions) can be performed only if certain conditions are met—for example, the game rules might stipulate that the target location for the move must be empty. In addition to the move conditions, there are constrains expressing the legal situations in the game. For instance, a move that is legal superficially may lead to a forbidden situation in the game. Consider the missionaries and cannibals problem. A legal move in terms of transportation may result in cannibalization of missionaries (i.e., a majority of cannibals at one shore). We refer to such situations as constraint violations or *contradictions* in the game.

We have implemented the board-game method using the chronological-backtracking method. A T-function adapted for board games handles the generation of subsequent states in chronological backtracking. This T-function uses one rule set that describes moves, and a second rule set that detects contradictions in states. The move rule set determines various possible moves in a given situation. After the possible moves have been determined, the T-function makes a copy of the current state. Furthermore, the T-function makes the assumption that moves and actions can be performed by removal of pieces from a location and addition of the same pieces to another location. The method implements this behavior by

```
(defrule hanoi-1  ; Every disk can be moved on a larger disk or on an empty peg.
        (position ?s ?piece&:(numberp ?piece) ?location 1)
        (value ?s ?new-location ?x&:(numberp ?x)&:(< ?piece ?x)|FALSE $?)
        =>
        (assert (action ?s ?piece ?location ?new-location)))

(deffunction hanoi ()
   (reset)
   (bind ?*locations* (mv-append v1 v2 v3))
   (bind ?*pieces* (mv-append 1 2 3))
   (make-instance [sI] of ?*state-class*)
   (set-slot-value [sI] v1 ?*pieces*)
   (send [sI] init-state)
   (make-instance [sg] of ?*state-class*)
   (set-slot-value [sg] v3 ?*pieces*)
   (print-result (solve (mv-append [sI]))))
```

Figure 6: The definition of the Towers of Hanoi problem for the board-game method.

calling a function that removes pieces from a location (state) variable, and by calling another function that adds the pieces to the target location variable. The T-function produces a list of subsequent states, which is checked by the contradiction rules for any illegal states. After illegal states have been removed, the remaining states optionally may be ranked according domain-specific heuristics. A third rule set can be used to encode such heuristics for preferred states (i.e., states close to the goal state in some sense). In addition to the T-function, several utility constructs are required to implement fully the board-game method as chronological backtracking. For instance, the board-game class is a subclass of the state class, and a generic equality function is defined for instances of the board-game class.

Note that certain classical AI problems—such as the frame and ramification problems—do not arise here, because we make strong assumptions on the nature of a move. For example, the moves have no side effects other than moving pieces between locations. Also, the consequences of a move are well defined.

We have used to board-game method to implement several classical toy problems. Figure 6 shows the definition of the Towers of Hanoi problem as a board-game problem. The definition of the rule `hanoi-1` and the function `hanoi` are expressed in CLIPS, since our implementations of the board-game and chronological-backtracking methods are CLIPS based.

The rule `hanoi-1` states that each topmost disk can be moved onto a larger disk or onto an empty peg. The precondition of `hanoi-1` checks whether there is a piece (disk) at the top of a location (peg), and whether the value of the target location is a piece (disk number) larger than (that of) the piece moved. The action part of `hanoi-1` asserts an action to be performed in the CLIPS fact list.

Initially, all disks are located on the first peg (v_1). The function `hanoi` creates and initializes the initial state (S_I) of the game. Furthermore, `hanoi` creates and initializes the goal state (S_G), where all disks have been moved to the third peg (v_3). When the initialization required has been performed, `hanoi` calls the problem solver for chronological backtracking (`solve`) with S_I as argument, and prints out the result by calling the function `print-result`.

Although the board-game method seems suited for small toy games, we can use it to solve an unexpectedly wide range of problems. In fact, the board-game method can solve the Sisyphus room-assignment problem. If we view the room-assignment problem as a game where the persons are pieces, and the rooms are locations, we can define legal moves for persons between rooms (or the unassigned location). Initially, all persons are located outside the building (i.e., nobody is assigned a room). The goal is to bring all pieces (persons) inside the building under the room-assignment constraints; the goal predicate checks for an empty outside location. Contradictions occur, for instance, when a smoker and a nonsmoker are assigned to the same room (because the problem definition stipulated that smokers and nonsmokers should not be mixed). If a contradiction develops, the problem solver backtracks and attempts another series

of moves. The result of this algorithm is the goal state in which persons have been assigned correctly to rooms.

Figure 7 shows the code required for defining the room-assignment problem in the board-game method. The moves are specified by the rule `sisyphus-1`. The rule essentially states that it is legal to move the first unassigned person into an available room that matches the person's role. The problem definition is backward compatible with the rules for person role and room properties in Figure 5, and the same knowledge-acquisition tool can still be used to acquire the rule set. The rule `sisyphus-2` queries the rule set in Figure 5 for unassigned persons. The rule `sisyphus-share-smoke-1` detects any contradiction where smokers and nonsmokers share the same room. The rule `sisyphus-goal-1` implements the goal predicate by testing whether the location representing unassigned is empty. Finally, the function `sisyphus` creates the initial state (S_I) and initializes the initial state with all the persons at the location unassigned. Thereafter, the `sisyphus` function calls the problem solver and prints the result.

The implementation of the Sisyphus room-assignment problem based on the board-game method is much more compact than is the version only using chronological backtracking. In fact, the essential part of the problem definition for the board-game method fits into one page of code (see Figure 7). Nevertheless, additional code for the mapping from person roles to room requirements is required.

In addition to the Towers of Hanoi and the Sisyphus problems, we have implemented the cannibals-and-missionaries problems and the farmer's dilemma—a variant of cannibals and missionaries where a farmer attempts to move a fox, a goat, and a cabbage across the river. Figure 8 illustrates the relationship between the implementations. We have used chronological backtracking to implement the Towers of Hanoi, farmer's dilemma, and Sisyphus problems—that is, these problems have been specified as T-functions. Furthermore, we have implemented the board-game method by a T-function for chronological backtracking. This T-function is generic in the sense that the developer can instantiate it for board-game problems by specifying rules for moves and contradictory situations in the game. In turn, the Towers of Hanoi, farmer's dilemma, cannibals-and-missionaries, and Sisyphus problems have been specified as board-game methods.

Although the board-game method can be used to specify several of the most common AI toy problems, it has several drawbacks. For example, the current version of the board-game method handles only one-player games. Further, it is not clear whether the board-game method can solve real-world problems other than AI-world problems (e.g., Towers of Hanoi and blocks world). The specification of moves and contradictions can be difficult to implement in a more systematic manner than rule sets (which are close to arbitrary expressions). Finally, problem definitions are difficult to debug, partly because they are extremely terse.

7 Results

Probably the most important factor that determines the usefulness of reusable problem-solving methods is the time required to adapt a method for the problem and to integrate that method with other methods in the expert system under development. We seek to develop a framework for configuration of problem-solving methods that minimizes the knowledge-engineering time required. Table 4 shows the numbers of lines of code that were required to implement various problems in the chronological-backtracking and board-game methods.[2] The implementation of the Sisyphus room-assignment problem required 278 lines of code, whereas the recursive algorithm (without utility functions) for chronological backtracking was implemented in 6 lines of code. In a sense, chronological backtracking is highly reusable, because it is general and simple. However, chronological backtracking requires substantial work to be reused for any interesting task. In this respect, chronological backtracking is inappropriate as a problem-solving method.

The board-game method is much better than chronological backtracking in this respect. The implementation of the Sisyphus problem as a board game required only 36 lines of code. The results for the other problems we tried are similar (see Table 4). However, these problems are well behaved, in the sense

[2]There are many problems associated with measuring the complexity of a program by counting the number of program lines. Nevertheless, the number of lines is one of the simplest and most intuitive measures we have. Moreover, since the programs analyzed are relatively small, the difference between various metrics is of less importance.

```
(defrule sisyphus-1                ; Move a person to a room
        (position ?s ?person unassigned 1)
        (room-query (state ?s) (size ?size1) (smoker nil) (central ?c1))
        (room ?room ?size2&:(eq ?size1 ?size2)|:(eq ?size1 nil)
                    ?c2&:(eq ?c1 ?c2)|:(eq ?c1 nil))
        (value ?s ?room&~unassigned ?mate)
        =>
        (assert (action ?s ?person unassigned ?room)))

(defrule sisyphus-2                ; Query the role knowledge base
        (position ?s ?person unassigned 1)
        =>
        (bind ?role (slot-value ?person role))
        (assert (query ?s ?person ?role)))

(defrule sisyphus-share-smoke-1    ; Detect smoke and sharing violations
        (value ?s ~unassigned ?p1&:(instancep ?p1) ?p2&:(instancep ?p2))
        (test (or (slot-value ?p1 single-office) (slot-value ?p2 single-office)
                  (neq (slot-value ?p1 smoker) (slot-value ?p2 smoker))))
        =>
        (assert (contradiction ?s)))

(defrule sisyphus-goal-1           ; The goal is to assign all personnel a room
        (value ?s unassigned FALSE)
        =>
        (assert (goal ?s)))

(deffunction sisyphus ()
  (reset)
  (bind ?*locations* (mv-append unassigned
                                (find-all-instances ((?r room)) TRUE)))
  (bind ?*pieces* (find-all-instances ((?r person)) TRUE))
  (do-for-all-instances ((?r room)) TRUE (assert (room ?r ?r:size ?r:central)))
  (make-instance [sI] of ?*state-class*)
  (set-slot-value [sI] unassigned ?*pieces*)
  (send [sI] init-state)
  (make-instance [sg] of ?*state-class*)
  (send (solve (mv-append [sI])) print))
```

Figure 7: The definition of the Sisyphus room-assignment problem for the board-game method.

Table 4: Lines of code required for various problem definitions for the chronological-backtracking and board-game methods.

problem	problem solver	lines of code
Towers of Hanoi	chronological backtracking	76
Farmer's dilemma	chronological backtracking	72
Sisyphus	chronological backtracking	278
Towers of Hanoi	board game	17
Farmer's dilemma	board game	31
Missionaries & cannibals	board game	32
Sisyphus	board game	39

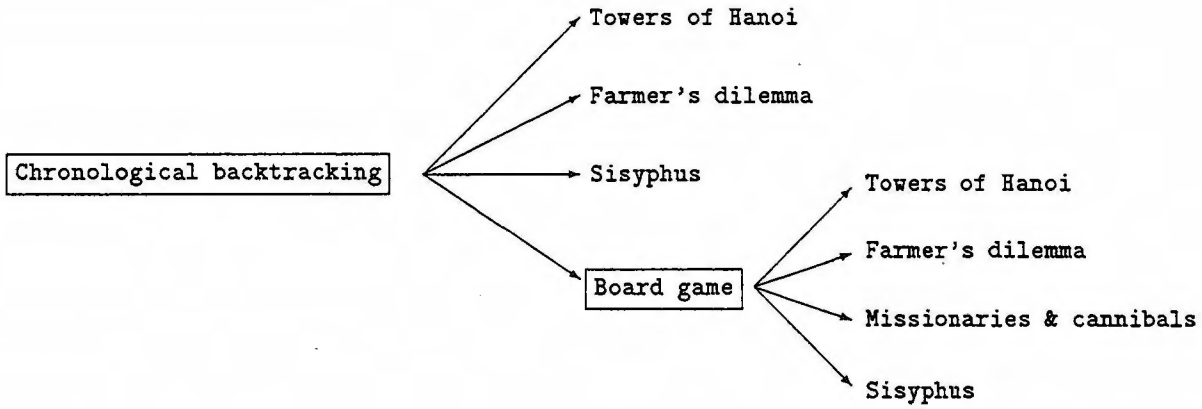


Figure 8: Programs implemented. The chronological backtracking method has been used to implement the standard problems (Towers of Hanoi, Farmer's dilemma, and Sisyphus), as well as to realize the board game method. In turn, the board-game method has been used to reimplement the three standard problems as well as the Missionaries and cannibals problem.

that they can be modeled easily as board-game problems. In comparison to the size of the reusable component in chronological backtracking, the generic T-function that implements the board-game method in the chronological-backtracking method required about 160 lines of code, with an additional 80 lines for utility functions. Thus, the level of reuse (at least, in terms of lines) is significantly larger when these problems are implemented with the board-game method.

The number of lines of code required for each problem does not, however, give a complete picture of the utility of reusable methods. Just because a problem definition can be expressed in a compact way does not mean that can it be *easily* modeled and described. What we want to measure is not the number of lines required to implement a problem, but rather the amount of modeling work needed. Unfortunately, it is difficult to measure quantitatively the amount of such work, because the modeling is largely a cognitive process. For instance, the effectiveness of a problem-solving method in knowledge-engineering terms depends on at what stage and at what rate the developer gain insight into how the problem can be modeled. When the developer realizes that the Sisyphus room-assignment problem can be viewed as a board game, the board-game method is elevated from irrelevant to highly useful.

An important question is whether the board-game method allows us to implement board games faster than chronological-backtracking does. The distance between the problem in question and the potential problem solver is much less when the board-game method is used for the problems intended. This fact, like our examination of the modeling required, indicates that the board-game method indeed makes it easier to implement native board games and problems that can be expressed as board games.

8 Discussion

We have examined how several example problems can be solved by generic problem-solving methods, particularly the chronological-backtracking and board-game methods. There are two crucial issues for reusable problem-solving methods: (1) What are the contributions of reusable methods in the development process? and (2) How can methods be combined into working knowledge-based systems? In this study, we have examined the first issue, and we have discussed the configuration of problem-solving methods. In a sense, chronological backtracking was a disappointment as a problem-solving method. Although the method is general, it does not provide much support for the developer, since it required an extensive T-function for its configuration. In contrast, the board-game method provides much more support than chronological backtracking does, but we still do not know how to map problems that are not native board games (e.g., Sisyphus) into board-game definitions methodically. Also, we do not have systematic means for determining whether a given problem is—or can be viewed as—a board game. In Section 8.1, we discuss

the configuration of problem-solving methods; in Section 8.2 we examine the combination of ontologies, tasks, and methods.

8.1 The Anatomy of Problem-Solving Methods

Using the board-game method for solving the Towers of Hanoi problem by representing the problem as a board game is not necessarily the most natural way to represent this task. One distinction we found useful when analyzing and classifying tasks is whether a task is *operator oriented* or *configuration oriented.* In the operator-oriented case, operators for transferring a state into subsequent states are inherent in the task definition. In the configuration-oriented case, operators are nonexistent or irrelevant; the properties of the goal configuration are the main attributes of the solution, not the way of arriving at the goal configuration. From this point of view, Sisyphus is configuration oriented, while the other board-game tasks we have examined are operator oriented (e.g., Towers of Hanoi and Missionaries and cannibals). The Sisyphus task has to be mapped into the class of operator-oriented tasks (e.g., by defining explicitly an imaginary *move* operator) before it can be approached as a board game. If our intent is to solve simply the specific Sisyphus room-assignment problem, it is more reasonable to select a constraint-satisfaction problem-solving method better suited for configuration-oriented tasks than to select the board-game method. Note, however, that the board-game method also can solve the slightly different task of rearranging persons with a minimum amount of moves when some condition changes (e.g., someone starts smoking)—a task that would be beyond simple constraint satisfaction.

An important question for researchers exploring reusable components for knowledge-based systems is how to *describe* reusable problem-solving methods. In PROTÉGÉ-II, the descriptions of tasks and methods include their input and output. Nevertheless, tasks and methods have other properties that are crucial for the method-selection and method-configuration processes. Methods might require expressions, rules, and functions as part of their configuration if they are to work properly. For instance, chronological backtracking requires a T-function—that is, a generator for subsequent states.

The role of expressions, rules, and functions in the configuration of problem-solving methods has a substantial influence on how knowledge-based systems are designed and implemented. Initially, we thought of the T-function as *input* to the chronological-backtracking method. Such input is complex, however, and we cannot expect users to provide arbitrary functions and expressions as input. If functions and similar programming elements are to be considered as input to methods, they have to be provided by the developer (knowledge engineer) at method-configuration time. An alternative to viewing functions as input is to view them as submethods accomplishing subtasks. (Functions have input and output just as methods in PROTÉGÉ-II do; see Section 2.) This perspective allows the developers to select methods implementing various functions from a library, and, possibly, to decompose these methods further. For example, the T-function implementing the board-game method in the chronological-backtracking method is a submethod of chronological backtracking. Furthermore, the move and contradiction rules implementing the various problems are, in a way, submethods of the board-game T-function.

A question related to that of the role of functions and similar constructs is what the specification language for problems should be. In the chronological-backtracking method, the developer specifies problems to the method by writing a T-function in a language such as CLIPS. The board-game method, on the other hand, requires that the problem be specified as rules for moves and contradictions. The art of creating a good problem-solving method comprises finding methods to characterize a set of problems in a uniform way that can also be understood easily. In many ways, chronological backtracking is too general to be a good problem-solving method, because the T-function does not impose any structure on problem specification or provide any guidance for the developer. The board-game method is much more effective in this respect, since it is more specific than is chronological backtracking. A problem-solving method even more specific than the board-game method may be too restricted in terms of the number of potential problems it can handle. Thus, we are considering problem-solving methods that have a grain size similar to that of the board-game method.

Knowledge-acquisition analysis is a crucial activity that the developer must take into account when considering reusable components for knowledge-based systems. An important factor in the selection of an appropriate problem-solving method for a task is the available expertise. If the expertise required to solve a problem in a certain way cannot be acquired, the method is inadequate. Often, knowledge-acquisition

analysis is required before problem-solving methods for accomplishing the tasks can be selected.

8.2 Combination of Ontologies, Tasks, and Methods

Configuration of problem-solving methods may involve various forms of data description (e.g., subclassing) and procedural specification (e.g., T-functions). Furthermore, methods typically require expertise to perform their tasks. In PROTÉGÉ-II, we distinguish three levels of method configuration. First, methods can be configured by the developer using the PROTÉGÉ-II tool. For example, the developer must write T-functions for the chronological-backtracking method and move rules for the board-game method. Second, methods can be instantiated by domain knowledge entered by experts in a knowledge-acquisition tool generated by PROTÉGÉ-II. For instance, experts can enter heuristics for the Towers of Hanoi problem, and can enter tables translating personnel roles to room requirements for the Sisyphus problem. Third, the end user and the other methods can provide input data to an activated method during run time. Likewise, an activated method can produce data for other methods and for output from the system.

An important factor influencing success in reuse of components for knowledge-based systems is the ability to *combine* reusable data and operations on those data—that is, ontologies and problem-solving methods. Most of the current approaches to sharable ontologies and reusable methods do not provide adequate support for modeling how these entities should function together. Classically, data and programs are considered as duals of each other, and the key to creating programs is to combine them (Wirth, 1976). Since knowledge-based systems are also programs and, thus, obey the same rules as do other programs, the combination of ontologies and methods is vital. We do not claim to have a solution to the problem of combining sharable ontologies and reusable methods, but we do recognize the difficulty of combining them and the challenge of bridging the two approaches. In particular, we are working on techniques for describing reusable problem-solving methods and their ontologies (Walther, Eriksson & Musen, 1992).

In addition to the combination of ontologies and methods—and the corresponding domain conceptualization and task analysis—the developer must analyze the needs and possibilities for knowledge acquisition from the sources available, and must produce a plan for acquiring the relevant knowledge. This process may involve the selection of appropriate knowledge-elicitation tools, the configuration of a knowledge-acquisition workbench, and the development of a domain-specific knowledge-acquisition tool.

Several approaches to knowledge-acquisition tools for problem-solving methods have associated a method-specific knowledge-acquisition tool with each method. For instance, MOLE (Eshelman *et al.*, 1987) supports knowledge acquisition for the cover-and-differentiate method (classification), SALT (Marcus & McDermott, 1989) supports knowledge acquisition for the propose-and-revise method (design), and PROTÉGÉ (Musen, 1989) supports knowledge acquisition for skeletal-plan refinement. PROTÉGÉ differs from MOLE and SALT in that it is a meta-tool that allows the developer to instantiate knowledge-acquisition tools for specific domains. PROTÉGÉ-II, however, is designed to relax such one-to-one mappings between knowledge-acquisition tools and problem-solvers and to generate domain-specific knowledge-acquisition tools that span several problem-solving methods used in an application system.

9 Conclusions

This work represents a bottom-up approach to examining reusable problem-solving methods. Rather than working on the architectural level, where we cannot automatically assume that reusable components will actually fit together at lower levels, we have descended to examining specific problems and implementations of problem-solving methods. In essence, this examination has revealed that the design of reusable component is an engineering challenge.

McDermott (1988) and other researchers distinguish between *general* and *role-limiting* methods for problem solving. Sometimes, the phrases *weak* and *strong* methods are used to denote general and specific methods, respectively. Weak methods make only weak assumptions about the task (i.e., a general method such as chronological backtracking). Strong methods make strong assumptions about the task (i.e., a specific method, such as the board-game method). We reject the terminology of weak and strong methods, since we find it counterintuitive and somewhat misleading due to the connotations of the words *strong* and *weak*.

The power of general methods is that they can solve a broad range of problems (albeit at the price of more configuration work than that required for role-limiting methods). Not surprisingly, our research indicates that we can indeed use general methods for many problems, but that the cost of doing so might be too high, especially when the amount of reused code is taken into account. (Note that many problem-solving methods are Turing-complete, and, in principle, can be used to solve any computable problem.) Nevertheless, general methods can be used to implement role-limiting methods (e.g., chronological backtracking can implement the board-game method). Furthermore, general methods do not constitute knowledge roles that can guide sufficiently the method configuration and the knowledge-acquisition analysis. Role-limiting methods, on the other hand, provide more structure and guidance than do general methods.

There is still the question whether role-limiting methods provide *sufficient* structure and guidance, especially for tool-supported method configuration. Our analysis of the board-game method suggests that even role-limiting methods might not be sufficient for configuring the method automatically—the task and knowledge-acquisition analyses are still human activities. For example, the board-game method requires rules defining moves of pieces and contradictory (illegal) positions in the game. In principle, the rules can be arbitrary from the method's point of view.

The strength of a problem-solving method is determined by the context in which that method is used. We advocate that the power of a method be determined relative to that method's usage in each application. For instance, chronological backtracking is a general method when applied to the Towers of Hanoi problem. Nevertheless, for tasks that use the notion of states and generators for subsequent states intrinsically (e.g., implementations of specialized problem solvers), chronological backtracking is a role-limiting method. The implementation of the board-game method in the chronological-backtracking method is an example of a case where chronological backtracking is a role-limiting method, since the board-game task deals with situations and subsequent situations. We suggest that the meaning of *general method* and *role-limiting method* be reinterpreted such that in a general method extensive method configuration is required relative to the *task in question,* and in a role-limiting method the method configuration for the task is straightforward.

Acknowledgments

This work has been supported in part by grants LM05157 and LM05208 from the National Library of Medicine, by grant HS06330 from the Agency for Health Care Policy and Research, by a gift from Digital Equipment Corporation, and by scholarships from the Swedish Institute, from the Fulbright Commission, and from Stanford University. Dr. Musen is recipient of National Science Foundation Young Investigator Award IRI-9257578.

We are grateful to Lyn Dupré for providing editorial assistance, and we thank Eckart Walther for his comments.

References

Boose, J. H. (1985). A knowledge acquisition program for expert systems based on personal construct psychology. *International Journal of Man–Machine Studies*, 23(5):495–525.

Clancey, W. J. (1985). Heuristic classification. *Artificial Intelligence*, 27(3):289–350.

Dean, T. (1988). Planning paradigms, in William Swartout, editor, DARPA Santa Cruz workshop on planning. *AI magazine*, 9(2):115–119.

Eriksson, H. (1991). *Meta-Tool Support for Knowledge Acquisition.* PhD thesis 244, Linköping University, Linköping, Sweden.

Eshelman, L., Ehret, D., McDermott, J. & Tan, M. (1987). MOLE: A tenacious knowledge-acquisition tool. *International Journal of Man–Machine Studies*, 26(1):41–54.

Firby, R. J. (1987). An investigation into reactive planning in complex domains. In *Proc. of AAAI-87*, pages 202–206.

Hale, R. (1987). Temporal logic programming. In Galton, A., editor, *Temporal Logics and Their Applications*, chapter 3, pages 91–119. Academic Press, London.

Kahn, G., Nowlan, S. & McDermott, J. (1985). Strategies for knowledge acquisition. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI*, 7(5):511–522.

Karbach, W., Linster, M. & Voß, A. (1990). Models, methods, roles and tasks: Many labels—one idea? *Knowledge Acquisition*, 2(4):279–299.

Klinker, G., Bhola, C., Dallemagne, G., Marques, D. & McDermott, J. (1991). Usable and reusable programming constructs. *Knowledge Acquisition*, 3(2):117–135.

Korf, R. E. (1985). Macro-operators: A weak method for learning. *Artificial Intelligence*, 26(1):35–77.

Linster, M. (1992). Sisyphus'92: Models of problem solving. Technical Report 630, GMD, St. Augustin, Germany.

Marcus, S. & McDermott, J. (1989). SALT: A knowledge acquisition language for propose-and-revise systems. *Artificial Intelligence*, 39(1):1–37.

McDermott, J. (1988). Preliminary steps toward a taxonomy of problem-solving methods. In Marcus, S., editor, *Automating Knowledge Acquisition for Expert Systems*, chapter 8, pages 225–256. Kluwer Academic Publishers, Norwell, MA.

Musen, M. A. (1989). *Automated Generation of Model-Based Knowledge-Acquisition Tools.* Morgan-Kaufmann, San Mateo, CA.

Musen, M. A. (in press). Dimensions of knowledge sharing and reuse. *Computers and Biomedical Research.*

NASA. Software Technology Branch, Lyndon B. Johnson Space Center, NASA (1991). *CLIPS Reference Manual*, Houston, TX.

Puerta, A. R., Tu, S. W. & Musen, M. A. (1992). Modeling tasks with mechanisms. Technical Report KSL-92-30, Knowledge Systems Laboratory, Stanford University, Stanford, CA.

Steels, L. (1990). Components of expertise. *AI Magazine*, 11(2):28–49.

Walther, E., Eriksson, H. & Musen, M. A. (1992). Plug-and-play: Construction of task-specific expert-system shells using sharable context ontologies. Technical Report KSL-92-40, Knowledge Systems Laboratory, Stanford University, Stanford, CA.

Wirth, N. (1976). *Algorithms + Data Structures = Programs.* Prentice-Hall, Englewood Cliffs, NJ.

An Analysis of Languages which Operationalize and Formalize KADS Models of Expertise

Dieter Fensel and Rudi Studer
Institut für Angewandte Informatik und Formale Beschreibungsverfahren
University of Karlsruhe, P.O. Box 6980, 7500 Karlsruhe, Germany
e-mail: {fensel,studer}@aifb.uni-karlsruhe.de

Abstract. The increasing number of languages which allow a formal and executable specification of the model of expertise at the knowledge level attempt to overcome shortcomings of life cycle oriented development processes of expert systems. Normally, due to the separation of the three activities knowledge acquisition, design, and implementation, the model of expertise is only described informally during knowledge acquisition. Changing this unsatisfactory situation is precisely the reason that languages like FORKADS, KARL, $(ML)^2$, MODEL-K, OMOS, and $K_{BS}SF$ were developed. This paper analyses three of them with respect to their modelling primitives, their semantics, their implementations, and their applications.

Introduction

One of the major results of the KADS-I project (cf. [BrW89, WSB92]) is the introduction of a life cycle oriented methodology for developing expert systems including a proposal for a so-called model of expertise or conceptual model as the result of the analysis phase.[1] This conceptual model is based on the identification of different kinds of knowledge, which are reflected by distinguishing four different layers within the conceptual model. Accordingly, different kinds of modelling primitives have been proposed for capturing the different types of knowledge being represented on the different layers of a conceptual model. However, all these proposed modelling primitives have only been defined informally. Thus, they lack a clear semantics and do not provide a basis for formally analysing the modelled expertise or for evaluating it by executing the model.

As a consequence, in the last three years an increasing number of languages have been developed to overcome this shortcoming. Up to now, the following languages have been primarily discussed in the literature:

- FORKADS (cf. [Wet90, WeS91]),
- KARL (cf. [FAL91, AFL91b]),

1. In the paper we do not distinguish between the model of expertise and the conceptual model.

- $(ML)^2$ (cf. [AHS90, HaB92]),
- Model-K (cf. [KVS91, KaV92a, KaV92c]),
- OMOS (cf. [Lin92a, Lin92b]),
- and $K_{BS}SF$ (cf. [JoS92, VJS92]).

Essentially, all these languages can be assigned to three different classes:

- languages aiming at formalizing KADS conceptual models, e.g. $(ML)^2$, $K_{BS}SF$;
- languages aiming at operationalizing KADS conceptual models, e.g. MODEL-K, OMOS;
- languages aiming at formalizing and operationalizing KADS conceptual models, e.g. FORKADS and KARL.

In this paper we compare three of these languages, namely OMOS, KARL, and $(ML)^2$. Our comparison includes one representative of each of these language classes.

Our paper is organized as follows. The first section discusses why such languages are needed in the framework of a life cycle oriented methodology. Based on that analysis four criteria are identified in section two for comparing these different languages:

- given modelling primitives,
- availability of a formal semantics for these primitives,
- executability of the language, and
- applications which have been developed by using these languages.

In section three, OMOS, KARL, and $(ML)^2$ are analysed in detail with respect to these criteria. The last chapter summarizes this analysis and makes the first step toward a meaningful comparison of the different languages.

1. Why all these stuff? Or how to overcome the pitfalls of life cycle oriented development methods.

Originally, knowledge acquisition was regarded mainly as a knowledge transfer process. The knowledge was viewed as an already given and fixed entity which should be elicited by the knowledge engineer or, better, by an automated knowledge acquisition tool. The problems (cf. [Ber87]) arising in its course are well known:

- There is an insufficient understanding of experts´ cognitive abilities to solve problems in their field.
- Human expertise is based on tacit knowledge and skill.
- Knowledge often is hardly to formalize and it is highly context sensitive.
- Experts rationalize their behavior instead of giving their real reasons.
- Experts are experts in solving problems, not in explaining their solutions.

These problems are indications of a significant feature of knowledge acquisition. Knowledge acquisition is not a process of transferring already given knowledge into a running computer program, but rather a process of knowledge creation. A model of expertise is created during knowledge acquisition which differs from the cognitive foundations of the expertise as it exists in the experts´ ability to solve problems (cf. [Mor87]). Every modelling process and, therefore, knowledge acquisition has some uncomfortable features: it is iterative, infinite, approximate, and faulty.

An early answer to this cyclic character of knowledge acquisition was rapid prototyping. ***Rapid Prototyping*** (cf. [Bra89]) means to immediately implement the acquired knowledge in a computer program. The running system may then be used to evaluate the gathered knowledge. This way of constructing a system reflects the above mentioned cyclic nature of the modelling proc-

ess. The running system immediately gives feedback to the knowledge acquisition in order to revise or complete it. In the past this method was therefore applied in most cases in order to construct an expert system. Nevertheless, this method has some well known disadvantages (cf. [AFS90]):

- The knowledge engineer who implements the system has to do a lot of jobs at the same time. He has to analyse the given information, design the system, implement the system, and evaluate the system.
- Different kinds of knowledge have to be considered and implemented simultaneously and are therefore mixed.
- The running system is the only documentation of the expertise.

The problems of the rapid prototyping approach arise because the running computer program is the only representation medium for the created model of the expertise. Implementational details become mixed with aspects of the conceptual model. To overcome these kinds of problems, the separation of ***symbol level*** and ***knowledge level*** was introduced by [New82]. On the knowledge level the model of expertise is described independent from its realization as an implemented program. This level can be used to describe an expert system's problem-solving method (cf. [Cla85]).[1]

The differentiation between knowledge and symbol level leads to an analogous distinction of analysis and design for the development process of a knowledge-based system. Speaking in terms of software engineering, a knowledge level description is the specification of the system independent of its design and implementation.[2] A life-cycle oriented development process[3] with different phases and defined milestones becomes possible. In the knowledge acquisition phase a model of the expertise has to be created. In the design phase additional issues are considered in order to build an efficient system.

When this paradigm is applied, the building process of experts systems becomes structured in a way which justifies the term knowledge ***engineering***. Besides its advantages, life cycle oriented process models suffer from a significant problem. They require fully elaborated documents as results of every phase. These documents should be correct, complete, consistent, and unequivocal. But the document of the first phase, the model of expertise, is normally an informal or semiformal one. It is neither possible to analyse it automatically nor to execute it. Therefore this important document generally contains a lot of inconsistencies and ambiguities.

There are three possible answers to this shortcoming without losing the advantages of life-cycle oriented development process are possible:

- More adequate process models try to integrate the separation of different phases and steps into a cyclic and reversible process model. Examples in the software engineering area are the ***spiral model*** (cf. [Boe86, Boe88]) or ***evolutionary prototyping*** (cf. [Flo84, FRS89]).[4]
- Formal specification languages which should improve the result of the specification phase have been developed. In [AHS90] some of the advantages of such a description of a model of expertise are summarized:
  - It reduces the vagueness and ambiguity of descriptions in natural language.

---

1. This conceptual description of ***reusable*** problem-solving methods (cf. [BWS87]) independent from their realization as a program code appears to be one of the major impacts of this separation. Therefore, one of the key issues for the analysis of the languages is the extend to which a language represents generic inference knowledge and how it connects this knowledge with domain specific knowledge.

2. In regard to the different purposes of a specification in traditional software engineering and Artificial Intelligence see [AFS90].

3. The stagewise model (cf. [Ben56]) and the waterfall model (cf. [Roy70]) are early ones.

4. In [NeS92] the KEEP process model for knowledge engineering is introduced to allow an incremental development

- The formal description can help to attain a clearer understanding of the individual problem-solving steps and the entire problem-solving methods which support their reuse.
- A formalized specification can be mapped to an operational one which allows prototyping to support knowledge evaluation.
- A formalized specification allows symbolic execution to evaluate the modelled expertise (cf. [CSB82]) and enables checks for consistency and completeness.

- Operational specification languages are developed to improve the result of the specification phase. Prototyping can be used to evaluate the specified knowledge. In [Flo84] this kind of prototyping is called ***explorative prototyping***. Prototyping can be used to detect errors or to asses the modelled solution, i.e. the chosen problem-solving method.

2. How to analyse formal and/or operational knowledge specification languages?

It is easy to understand that several entities can only be analysed according to a defined criterion and that the definition of this criterion determines the result of their investigation. Which criteria are suitable for investigating specification languages used to specify knowledge at the knowledge level? Criteria which can be used to compare programming languages[1] seem hardly suitable due to the different purposes of the different language types. Programming languages should allow an effective and efficient realization of a modelled expertise at the symbol level and therefore deal with different aspects.

As stated above, there is need for languages which allow a formal and executable specification of models of expertise at the knowledge level. Therefore, three different kinds of questions can be posed to the languages (cf. [Bra85]):

- ***Epistemological level***: Does the language offer adequate modelling primitives to allow a specification at the knowledge level. With the four-layer KADS model of expertise, a framework for such languages is defined. Does a language fulfil this framework?
  - Which language primitives are offered at the domain, the inference, the task, and the strategic layer?
  - How are the different layers connected, i.e. how is the idea of a reusable and therefore generic inference layer integrated, i.e. how is a generic inference layer connected with a domain layer?
- ***Formal level***: Is the language intended as a tool to ***formalize*** a model of expertise or is it only an operationalization language? This is a question addressing the semantics of the available modelling primitives. Has the language a formal semantics or is the semantics only defined informally or by an interpreter? Are there tools which support checks for consistency, correctness or completeness? What is the expressive power?
- ***Operationalization*** Is the language intended as a tool to ***operationalize*** a model of expertise or is it only a formalization language? If it is intended as an operationalization language than the following questions arise: What is its expressive power? It is easy to offer a language with high expressive power, natural language, for example. But it will become very difficult to make it running on a computer. An operationalization language should support error detection and assessment of the modelled expertise. Are there any tools supporting these functions?
- If the language is intended as an operationalization and formalization, how are these two aspects related?

---

1. See [AnK92] for a comparison of Modula with C.

- An additional criterion is the reported applications of a language.[1]

The following investigation of the languages is organized according to the criteria given above.

3. The languages in more detail

A model of expertise according to the KADS-I project (cf. [BrW89, WSB92]) separates different kinds of knowledge and provides different epistemological primitives for each of these types of knowledge. The knowledge acquisition phase results in four different kinds of knowledge. As this proposal defines the common framework for all languages discussed above, a short sketch of the modelling primitives is given.

- The ***domain layer*** contains domain specific knowledge about objects, concepts, their features and their relationships. At this layer the objects and the terminology of the domain which the knowledge-based system should use are described.
- The ***inference layer*** contains knowledge about the used problem-solving method. This layer indicates which inferences are necessary within a problem-solving method. This knowledge is meta-knowledge because it describes the application of the domain knowledge making inferences. It consists of two kinds of elements: ***meta classes*** and ***knowledge sources***. Meta classes provide or collect input or output of knowledge sources. They specify the concepts or structures of the domain layer which are premises or conclusions of inferences. Inferences are described via knowledge sources. A meta class can be a conclusion of one and a premise of another knowledge source. An ***inference structure*** is a network which shows the dependencies between meta classes and knowledge sources.
- The ***task layer*** contains knowledge about the control flow of a problem-solving method for solving a specific task. This layer specifies when inferences are made.
- The ***strategic layer*** contains meta-knowledge about combination and selection of tasks in order to solve a problem adequately. As this layer is up to now only very weakly defined we will not discuss it further here.

On the one hand, all languages which are discussed above use KADS as a framework for the available language primitives. Therefore great similarities can be detected among them. On the other hand, because these modelling primitives only have an informal semantics, different interpretations arise when one tries to formalize them.

3.1 Operational Models of Problem-Solving (OMOS)

"Operational Models of Problem-Solving (OMOS)" (cf. [Lin92a, Lin92b]) is a language to operationalize KADS models of expertise. It was developed at the German National Research Center for Computer Science, GMD, in Bonn. It is implemented in BABYLON (cf. [CPV89]). BABYLON offers frames, rules, Prolog, and constraints and is implemented in Common Lisp.

The main point of OMOS is to extend the Role-Limiting Methods (cf. [Mar88]) or Method-to-Task approach (cf. [Mus89a, Mus89b]) by integrating them into a KADS framework. The Role-Limiting Methods approach offers shells with corresponding knowledge acquisition tools like MOLE, MORE or SALT (cf. [Mar88]). The shells contain a fixed data structure and an algorithm which works on it. The knowledge acquisition collapses into two activities.

- First, the proper shell must be chosen.
- Second, it must be filled up with cases, i.e. with assertional knowledge.

These approach is not meaningful, if no previously developed shell fits the chosen task.

---

1. Because the languages are meant as tools to support knowledge acquisition this is truly the most important criterion in the long run.

OMOS allows a bottom-up development of such problem-solving methods and offers some tools which support the acquisition of assertional knowledge.[1] A problem-solving process can be modelled by using role and value changes of given instances.

Besides extending existing expert system shell approaches OMOS is an example of the third kind of answer to the shortcomings of life-cycle oriented process models. It allows the operationalization of KADS model of expertise and therefore allows specified expertise to be evaluated using explorative prototyping.

3.1.1 The epistemological level

OMOS offers language primitives for the first three levels of a model of expertise, i.e. for the domain, the inference, and the task layer.

3.1.1.1 The domain layer primitives

On the ***domain layer*** terminological knowledge can be modelled by concept hierarchies and relationships between concepts. Frames available in Babylon are used to model concepts. These frames can form a hierarchy with multiple inheritance. Relationships between concepts are defined by a Prolog predicate with the corresponding concepts as argument types. Assertional knowledge can be modelled at the domain layer using instances of concepts and tuples which are elements of relations the predicates are interpreted with. This can be done extensionally by enumerating the elements or intensionally by using Prolog clauses.

3.1.1.2 The inference layer primitives

The inference layer gives inference actions (knowledge sources) and roles (meta classes) as modelling primitives. The functionality of an inference action is defined declaratively by stating

- the type of value-assignment it implements ("false", "initial", or "modify"),
- its type of role-assignment ("false", "multiple-assign", "substitute", or "transfer"),
- whether the inference action queries the user for missing knowledge ("true") or not ("false"),[2]
- the domain-layer relation it uses
- and the input, output, and control roles.

A compiler translates this definition into a forward rule-scheme. The inference which is drawn by an inference action is described by the domain relation it uses. Therefore a high amount of the problem-solving knowledge must be represented as tuples of domain relations. It is not possible to represent problem-solving knowledge generically way by describing the inference which is drawn by an inference action. Only the interaction of the inference actions can be represented generically through network of control, input, and output roles.

A role can be associated with domain layer instances and contain input, control information, and output of inference actions. The execution of an inference action can have two possible consequences:

- First it can change the attribute values of instances of its input role.
- Second it can assign instances of its input role to its output role.

Only instances of concepts can be associated with roles. Roles define a pointer-view of the domain layer, i.e. they do not contain virtual copies of domain elements. Therefore, recursive problem-solving processes cannot be modelled. In addition, the domain layer is used as a global

1. The development of predefined terminological structures and tools supporting the gathering of assertional knowledge is one of the main advantages of these approaches.

2. Because inference actions are typed whether they include user interaction or not, a part of the user/system interaction may be modelled by OMOS.

variable. All inference actions can read it and write on it and therefore they can communicate without having a common knowledge role, i.e. without depicting their interaction at the inference layer.

An inference action has exactly one input and one output role and several control roles. Only instances of the input role can be transformed to the output role. The members of a control role only control the inference but cannot be consumed by it.

The inference structure is implicitly defined by the connections of inference actions and roles and is not an additional explicit language primitive. Therefore, OMOS allows no hierarchical structuring and refinement of inference structures.

3.1.1.3 The task layer primitives

The names of the inference actions define primitive actions. These primitive actions can be combined by sequence, branch, and loops. No variables are given.

3.1.1.4 The primitives which connect the domain and inference layers

A role can be connected with a domain concept and an inference action must be connected with a domain relation. To define a mapping of an inference action, the corresponding domain relation, the correspondence between input and control roles of the inference action, and the arguments of the domain relation must be specified. A view definition which reconstructs domain expressions does not exist.

3.1.1.5 The primitives which connect the inference and task layers

Inference actions return a boolean value indicating whether a change took place or not. In addition, every role can be evaluated for a boolean value (false if the role is empty, true otherwise).

3.1.2 The Formal level

OMOS is not discussed as an formal specification language, but only as a operationalization language. Therefore, no formal semantics for the modelling primitives is given. There is only an executable semantics as defined by the BABYLON interpreter and the rule compiler.

3.1.3 The Implementational level

For a discussion of the expressive power, see section 3.1.5 Discussion.

OMOS is implemented using BABYLON. Therefore, it includes the environment of BABYLON. In addition, three tools for knowledge level analysis of OMOS specifications were developed:

- Coverage analysis of relations: Checks whether the set of tuples of a relation covers all possible tuples according to the argument definitions of the relation.
- Dead-end analysis: The domain concepts which are associated with input and control roles of an inference action are used to check the completeness of the domain relation which is associated with the inference action.
- Loop-Hole analysis: The domain relation associated with an inference action is used to test for completeness of the domain concepts which are associated with the input and control roles of the inference action.

These tools support the knowledge refinement step. The idea of tool support for knowledge refinement is extended to arbitrary problem-solving methods which can be modelled with the language primitives of OMOS. Tools like MOLE or SALT are only constructed for one fixed problem-solving method.

3.1.4 Applications

OMOS has been applied for a bunch of relevant problems with finite problem space:

- ***Planning***: OMOS is used to remodel parts of ONCOCIN (cf. [LiM92]). The modelled problem-solving method is skeletal-plan-refinement.
- ***Selection***: The clamping tool selection problem in mechanical engineering is modelled in OMOS (cf. [KLS91]).
- ***Assignment***: The so-called Office-Plan or Sisyphus Problem (searching for an assignment of employees to appropriate working places) is modelled in OMOS (cf. [Lin91]).

3.1.5 Discussion

The application of OMOS is limited to problems with finite problem space because no new elements of domain concepts or domain relations can be created during the problem-solving process. Only attribute values and roles for elements of domain concepts can change. Therefore, only problems having solutions requiring selection among a predefined set of alternatives can be solved.

Change in the value and role of elements of domain concepts are the only intermediate results of the problem-solving process. No traces of it are available to guide the further problem-solving process. It is therefore difficult to model a problem-solving process which investigates several alternative paths.Loops and dead ends in the problem-solving path become very dangerous.

OMOS offers restricted modelling primitives for representing problem-solving knowledge and its connection with domain knowledge. Every inference action has exactly one input and one output role. It is not possible to describe the inference drawn by an inference actions in a generic manner. An inference action cannot define a view on a domain layer, i.e. it cannot link several domain expressions to form a combined term. Therefore domain relations, i.e. domain specific tuple sets, contain a great amount of the problem-solving knowledge.

OMOS is not a formal specification language for modelling expertise because it has no formal semantics. BABYLON code must be read in order to read the modelled expertise. But it is an operational language, allowing quick explorative prototyping which allows the evaluation of the modelled expertise by a running system. It gives the epistemological primitives of KADS and allows the bottom-up modelling of executable problem-solving methods. In addition, it has some tools which support the knowledge refinement step. It is akin in spirit to the role-limiting methods, but extends them by allowing new problem-solving methods to be defined.

3.2 KARL

The "Knowledge Acquisition and Representation Language (KARL)" (cf. [FAL91, AFL91b]) is a language used to formalize and operationalize KADS models of expertise. It has been developed at the University of Karlsruhe, Institut für Angewandte Informatik und Formale Beschreibungsverfahren. A formal description of the expertise is automatically mapped to an operational one. This mapping and the resulting code is implemented in Prolog. KARL tries to utilize results known from software engineering and information system design. Both disciplines have a long tradition in developing and applying formal and executable specification languages (cf. [Alf90, Dub89, Kun89, LNO89, Say90, Zav91]). The domain layer of KARL applies ideas of semantic data models (cf. [BrR84, Che76]) and object-oriented data bases (cf. [Bee90]) to represent the static knowledge. The inference layer applies ideas of structured analysis (cf. [Ros77, You89]) and the task layer language is influenced by languages like RSL (cf. [Alf90]).

KARL has three main features:

- It allows a formal specification of the model of expertise so as to avoid ambiguities.

- This specification is automatically mapped to a operational one. It becomes possible to evaluate the specified expertise by explorative prototyping.
- Normally, formal specifications are hard to understand and complex specifications, acquired for real life systems, become difficult to survey. Graphical representations with defined semantics as well as hierarchical refinement and structuring primitives are therefore given.

3.2.1 The epistemological level

KARL gives language primitives for the first three levels of a model of expertise, i.e. for the domain, the inference, and the task layer. The static domain knowledge and elementary knowledge sources are described declaratively by Horn logic with stratified negation. The procedural control knowledge can be described by means of an ordinary procedural language.

3.2.1.1 The domain layer primitives

KARL gives concepts and relationships for modelling terminological domain knowledge. Both can be ordered in is-a hierarchies with multiple inheritance. In addition to concepts and relationships, KARL provides Horn logic extended by stratified negation ([Ull88]) to express intensional descriptions, i.e. sufficient conditions.[1] A more elaborated logical language is provided to formulate constraints, which are necessary conditions for concepts and relationships. These constraints model integrity conditions for the domain layer and are not used during the inference process.

Objects model real-world entities. KARL distinguishes between objects which model real-world entities and attributes which model features of such entities as object-oriented databases (cf. [Bee90]). An object and its attributes is represented by a fact or instance, i.e. by a predicate which only has constants as arguments.

3.2.1.2 The inference layer primitives

An elementary knowledge source is described declaratively by a set of logical clauses using the above discussed language. In contrast to Prolog, the evaluation of these clauses is ***set-oriented*** (cf. [Ull88, Llo87]). Not one but all instantiations of a predicate are computed. The tuple oriented evaluation strategy of Prolog or (ML)[2] only computes one instantiation which applies order information implicitly encoded in the order of the clauses. Therefore, control information specifying the order of execution can implicitly encoded at the inference layer. Experience with production rules[2] show that this can cause problems. The declarative dependencies of these rules was often used to implicitly encode control knowledge.[3]

Knowledge sources communicate via meta classes. A knowledge source can have several input and output meta classes. Meta classes are modelled as containers. A knowledge source reads the contents of its input meta classes and overwrites the contents of its output meta classes. If the connection to an output meta class is labelled by an "+" the achieved result is added to its contents. If the connection to an input meta class is labelled by an "-" the input which is read is deleted from it.

A subset of knowledge sources and connected meta classes can be arranged to form one composed knowledge source, vice versa a knowledge source can be refined into a network of knowledge sources and meta classes. These composed knowledge sources can be used as elementary

---

1. These restrictions guarantee a finite and unequivocal model as semantics and allow for an efficient evaluation.
2. Experience with XCON (cf. [SBJ87, BaS89]) show that serious problems arise if the control flow is implicitly specified
3. If only one wants only one and not all instantiations are required, KARL forces the definition of an artificial order relation for the possible instantiations.

elements to specify further inference steps. Therefore, the inference processes can be described at several levels of granularity. The main advantages are:

- Modelling complex systems in one inference structure will result in an unreadable inference layer.
- It is possible to define several views on a system or a problem-solving method. Every view shows it at a different level of granularity.
- Problem-solving methods chosen out of a library (cf. [BWS87]) can be adapted to an actual task or domain by introducing an additional level of refinement.
- The top levels of inference structures which are developed bottom-up can be reused for similar tasks (cf. [LHS92]).
- If a system requires the cooperation of several predefined problem-solving methods, any of these methods can be represented by a composed knowledge source in one inference structure. The representation of the interactions of the methods is not confused with their internal structure.

The semantics of a composed knowledge source is defined by its elementary units and the control flow between them which is defined at the task layer. The refinement of a knowledge source can stop as soon as no further control knowledge has to be represented at the task layer and the behaviour of the knowledge source can be described totally declaratively.

Constraints can be used to describe pre- and post conditions of composed knowledge sources.

3.2.1.3 The task layer primitives

The names of the elementary knowledge sources define primitive actions. The names of the combined knowledge sources define subtasks. These names can be combined by sequence, branch, and loops. Primitive actions and subtasks return boolean values which can be stored in variables.

3.2.1.4 The primitives which connect the domain and inference layers

KARL offers a connection mechanism for both layers which is more powerful than OMOS and less than $(ML)^2$. An elementary knowledge source or a meta class can define a view on the domain layer using the above described extended Horn logic. A meta class need not correspond to a predefined domain concept and a knowledge source need not correspond to a predefined domain relationship. Several domain expressions can be combined to form a new expression by means of a view definition using Horn clauses.

3.2.1.5 The primitives which connect the inference and task layers

In the current version of KARL, knowledge sources return a boolean value.

3.2.1.6 Graphical representation

Normally, formal descriptions in Horn logic, BABYLON code, or nonstandard logic as proposed by $(ML)^2$ are difficult to understand. Formal specifications are therefore not a adequate tool for communicating with experts or system users. KARL offers graphical representations of most modelling primitives to improve their ability to be understood. Every graphical symbol has a defined meaning given by the semantics of the corresponding language primitive:

- Enhanced-Entity-Relationship diagrams for the domain layer,
- levelled data flow diagrams for the inference layer, and
- program flow diagrams for the task layer.

Specifications of real-world systems become unintelligible due to their complexity. Therefore all three graphical representations include hierarchical refinement to allow them to represent the

system at different levels of granularity.

3.2.2 The Formal level

The logical language used to describe the declarative parts of the expertise have a perfect Herbrand model semantics (cf. [Llo87, Ull88]). The constraints check this model for correctness. The procedural knowledge is represented by a procedural language which, up until now, did not have a formal semantics until now.[1]

3.2.3 The Implementational level

The expressive power of the declarative part of the language is similar to function-free Horn logic extended by negation.[2] The expressive power of the declarative and procedural part is Turing complete. A KARL specification is mapped to Quintus Prolog to make it executable. The mapping is implemented in Quintus Prolog, too. Currently, asides from some debugging facilities no further tools exist to support knowledge evaluation. Because of the operational semantics of Prolog, i.e. its incompleteness (cf. [Llo87]), the semantics of the prototype interpreter differs from the formal one. A new interpreter is being developed which computes the perfect Herbrand model as the semantics of a set of clauses (cf. [Ull88]).

3.2.4 Applications

KARL was applied to the following problems:

- ***Assignment***: The so-called Office-Plan or Sisyphus Problem (searching for an assignment of employees to appropriate work places) (cf. [AFL91a, AFL92b]), the configuration of devices to minimize pollution (cf. [LHS92]), and a further design problem (cf. [LFA92]) have been modelled in KARL.
- ***Selection***: The selection of scheduling algorithms is modelled in KARL (cf. [KFG92]).
- The problem-solving method ***Cover and Differentiate*** is modelled in KARL (cf. [Ang92]).

3.2.5 Discussion

When modelling the selection of scheduling algorithms it becomes apparent that there is a need for extended domain layer constructs. There was a need to model sets containing sets. At the inference layer, meta classes only contain domain instances and no concepts or relationships. Modelling reasoning about concepts or relationships would require meta reasoning as proposed by $(ML)^2$.

Currently, KARL does not offer a strategic layer. Modelling a complex control knowledge with KARL would result in a complex task layer. All control knowledge must be hard-wired and there is no support for depicting it in a reusable manner.

Currently, the formal semantics differs from the semantics of the prototype interpreter.

KARL is a language which allows to formalize and operationalize a model of expertise in an adequate and unequivocal manner. It offers explorative prototyping to evaluate a specified model. But up to now nothing has been said about support for the further phases, like design or implementation. There is no support for preserving the conceptual structure of the modelled expertise during these steps and for the resulting system.

---

1. This has changed but has not been published.

2. Because of Prologs top-down evaluation of clauses, the semantics of the negation differs from the formal semantics. This will be changed by the next KARL interpreter.

3.3 $(ML)^2$

$(ML)^2$ (cf. [WAS89, AHS90, HBA91, HaB92]) is a language to formalize KADS models of expertise. It was developed in the course of the ESPRIT Projects 3178 "REFLECT" and 5248 "KADS-II" and a bilateral research project of the Netherlands Energy Research Foundation ECN and the University of Amsterdam. Although a subset of $(ML)^2$ can be operationalized to allow explorative prototyping (cf. [THR91]) it is mainly introduced as a formalization language allowing the unequivocal formulation of models of expertise. It offers several embedded types of logic to declaratively describe a model of expertise. Modularized order-sorted first-order logic is used to describe a domain layer. The inference layer is modelled by meta-logic and the task layer by Quantified Dynamic logic.

The significant feature of $(ML)^2$ is its uniform application of logic for the different layers of expertise, especially for the representation of the task layer. The application of meta logic to represent knowledge at the inference layer comes close to the spirit of this layer. It describes the application of domain specific knowledge for the problem-solving process and therefore it is a meta layer in relation to the domain layer.

3.3.1 The epistemological level

$(ML)^2$ offers language primitives for the first three levels of a model of expertise, i.e. for the domain, the inference, and the task layers.

3.3.1.1 The domain layer primitives

The sublanguage of $(ML)^2$ used to model a domain layer is order-sorted first-order logic extended by modularisation. Concepts are modelled by constants, and sorts can be used to model classes of such constants. Sorts can be arranged in an is-a relation. Relationships between concepts are modelled by predicates. Attributes of concepts are modelled by functions. Arbitrary first-order theories can be used to specify the defined relationships. The specification of a domain layer can be divided into several modules. These modules, i.e. subtheories, can be combined by a union operator. Such a module or theory T:

- defines a signature, i.e. sorts, constants, functions, and predicates
- imports other theories
- defines axioms, i.e. logical formulas.

3.3.1.2 The inference layer primitives

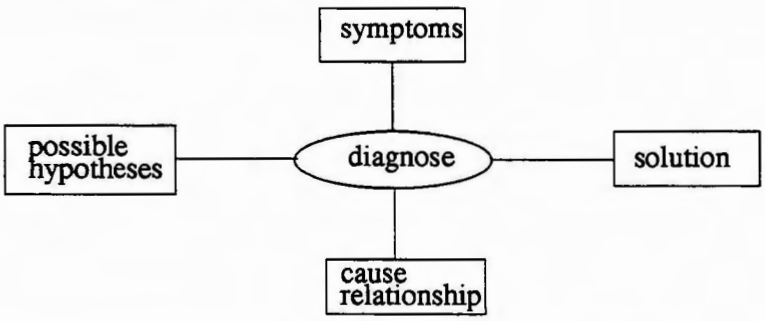


Figure 1. A small inference structure in $(ML)^2$

$(ML)^2$ gives primitive inference actions, knowledge roles, and a linkage of both as language

primitives at the inference layer. A primitive inference action (pia) could have several input roles, but has exactly one output role. Every pia and every role is described by a theory similar to domain layer theories. The theory which describes a role defines the mapping of domain layer terms on terms used at the inference layer. A theory describing a primitive inference action imports all theories of its input or output roles. Its axioms describe the inference drawn by the pia using the terms of the input and output roles. The figure 1, "A small inference structure in (ML)2", will be used to illustrate an inference layer modelled in $(ML)^2$.

The primitive inference action (pia) "diagnose" will diagnose a "solution" using "symptoms", "possible hypotheses", and a "cause relationship" as input. Notice that the "cause relationship" would be modelled as a meta relation in KARL whereas in $(ML)^2$ it becomes an input role. This pia "diagnose" would be modelled by a predicate:

$pia_{diagnose}$(symptoms(X), possible_hypotheses(Y), cause_relationship(symptoms(X), possible_hypotheses(Y)), solution(Y))

To describe the inference, one can use a set of clauses with the above given predicate or intermediate predicates as their conclusions and the input roles or intermediate predicates as premises. The whole description of the pia would be:

theory *diagnose*
  use symptoms, possible_hypotheses, cause_relationships
  signature
    predicates $pia_{diagnose}$
  variables X, Y
  axioms
    $input_{symptoms}$(symptoms(X)) &
    $input_{possible_hypotheses}$(possible_hypotheses(Y)) &
    $input_{cause_relationship}$(cause_relationship(symptoms(X), possible_hypotheses(Y)) -->
    $pia_{diagnose}$(symptoms(X), possible_hypotheses(Y), cause_relationship(symptoms(X), possible_hypotheses(Y)), solution(Y))

Hierarchical refinement of inference structures would become possible using the theories describing a pia but it is not discussed by the authors of $(ML)^2$.

3.3.1.3 The task layer primitives

Quantified Dynamic Logic is used to specify dynamic control at the task layer. Every predicate specifying a pia at the inference layer is regarded as an elementary program statement and the roles are used as input and output parameter of such programs. For every such elementary program a history variable V_{pia} is defined which stores the input-output pairs for every execution step. Four types of queries are available for such programs which are discussed for the pia "diagnose":

- ***has-solution***-$pia_{diagnose}(I_1, I_2, I_3, O)$: Checks whether the predicate "$pia_{diagnose}$" is evaluated to be true applying the axioms of its theory. "I_i" and "O" can be variables or instantiations, i.e. constants, for which the predicate is proved.
- ***old-solution***-$pia_{diagnose}(I_1, I_2, I_3, O)$: Asks whether "$(I_1, I_2, I_3, O)$" matches with an input-output pair stored in $V_{diagnose}$.
- ***more-solution***-$pia_{diagnose}(I_1, I_2, I_3, O)$: Asks whether a solution "(I_1, I_2, I_3, O)" exists which does not match with an input-output pair stored in $V_{diagnose}$.
- ***give-solution***-$pia_{diagnose}(I_1, I_2, I_3, O)$: This program first checks to see if a previously uncomputed solution exists, and if so, stores it in the corresponding Variable "$V_{diagnose}$". This language construct can be used to define a single step of a program. These single steps can be combined by sequence, branch, and loop and arranged to form subtasks. No recursion is available.

In KARL, for example, the meta classes work as containers holding the actual results of the knowledge sources. In $(ML)^2$ this task is performed by the history variables "V_{pia}". In addition, these variables do not store one state of the problem-solving process, but ***all*** states of the problem-solving process.

$(ML)^2$ uses a tuple oriented evaluation strategy of clauses similar to Prolog. Only one instantiation of a predicate is computed instead of the whole model of a formulae. In the case of $(ML)^2$ this is used to present different control flows at the task layer. One can distinguish whether one solution or all solutions of an inference action become computed.

3.3.1.4 The primitives which connect the domain with inference and task layers

The inference and task layers are modelled as a meta language. The domain layer is modelled as an object language. Both are connected by lift definitions and reflection rules.

At the inference layer, for every knowledge role which is connected to the domain layer, a theory and a lift operation are defined. The lift operator maps terms of the domain layer to variable-free expressions at the inference layer. A lift operator maps domain expressions in general terms used by a generic problem-solving method. In addition, rewrite rules can be applied to construct new expressions out of the domain terms. This is similar to the view definition of KARL, for example.Beyond KARL, not only tuples of domain instances but arbitrary domain terms, predicates or clauses, for example, can be mapped on terms at the meta layer.

At the task layer, "up-to-down" and "down-to-up" connections of domain and inference/task layer can be defined. This is done by applying the two reflection rules:

$$\frac{\phi \in O}{\vdash_{M} ask_{\in}(O,\lceil\phi\rceil)}\ \text{(ask: "down-to-up")} \qquad \frac{\vdash_{M} tell(O,\lceil\phi\rceil)}{\phi \in O}\ \text{(tell: "up-to-down")}$$

M = meta theory
O = object theory

"ask" allows the derivation of "ϕ " if "ϕ " is an axiom of the domain layer. "tell" stores "ϕ " as an axiom at the domain layer after "ϕ " was derived at the inference layer. Notice that both reflection rules distinguish between an axiom of a theory and a derived conclusion of a theory. Only existing axioms of a domain layer are connected down-to-up by "ask" to the inference layer. On the one hand, this is close in spirit to KADS. Only the inferences are drawn which are specified at the inference layer. On the other hand, an intensional descriptions of domain layer relations is not possible.[1] KARL offers the possibility for describing domain relations using clauses, which is considerable easier than being forced to describe all domain knowledge extensionally. These descriptions are evaluated independently from the chosen problem-solving method because they are short-cuts in describing domain knowledge.

3.3.1.5 The primitives which connect the inference and task layers

The predicates which characterize a pia return a boolean value and a variable binding. Because the task layer is not modelled as a meta layer to the inference layer but as an extension of it, no reflection rules are necessary.

---

1. According to private communication, this has currently changed and $(ML)^2$ will offer "down-to-up" connection for all derivable axioms.

3.3.2 The formal level

$(ML)^2$ offers three kinds of logical languages to describe models of expertise (cf. [BaA92])

- ***First-order logic*** at the domain layer. The use of order-sorted first-order logic and modularisation do not extend its expressive power. Such a language is semi-decidable and generally, it is not possible to compute a model efficiently.
- ***Meta-logic*** at the inference layer. Making statements about predicates, "transitive(p)", for example, would require second-order logic. Alternatively, the theory can be divided into two disjunct parts, an object theory and a meta theory. This allows to reflect about predicates of the object theory without necessarily extending the expressive power from first-order logic to second-order logic.
- ***Quantified Dynamic logic*** at the task and strategic layer. It is a modal extension of first-order logic. A state is characterized by the values of all variables and the true values of atomic formulas. History variables are defined which store all states during problem solving.

In [BaA92] TheME is described which supports the formalization of models of expertise using $(ML)^2$. It allows a graphical representation of $(ML)^2$ expressions and, for example, checks for syntactical correctness but it does not help to check the semantics of the modelled expertise.

3.3.3 The implementational level

In [THR91] a prototype interpreter $Si(ML)^2$ for a subset of $(ML)^2$ is discussed. The main restriction concerns the domain layer. It is restricted to Horn logic and negation is interpreted as negation as failure. Function symbols are not interpreted. Besides the executability of a limited subset of $(ML)^2$ no further tools for knowledge evaluation are given. $Si(ML)^2$ operationalizes an early version of $(ML)^2$ which has a different kind of task layer as discussed in the paper.

3.3.4 Applications

$(ML)^2$ has been applied in a dozen formalizations of models of expertise (cf. [HaB92]), e.g., for making predictions about physical systems by means of qualitative reasoning (cf. [BRW90]). The problem-solving methods ***Cover and Differentiate*** (cf. [SWA92]) and ***Heuristic Classification*** (cf. [AHS90, SWA92]) are formalized in $(ML)^2$.

3.3.5 Discussion

$(ML)^2$ applies different types of logic for formalizing a model of expertise. First-order logic, meta-logic and Quantified Dynamic logic are means for the different layers.

- $(ML)^2$ is the only language which offers a logical description for the control knowledge at the task layer.
- The application of meta logic to represent knowledge at the inference layer comes close to the meaning of this layer.

In addition to these advantages, there are still some problems with $(ML)^2$.

On the one hand, $(ML)^2$ offers a powerful languages for modelling domain knowledge. On the other hand, this language is only semi-decidable and it is not possible to evaluate it efficiently. It is no surprise that the operationalized subset of $(ML)^2$ is restricted to Horn logic.

$(ML)^2$ does not distinguish between objects and concepts. Both can be modelled using constants, and classes of such concepts can be modelled using sorts. To model concepts as constants

and classes of concepts as sorts violates an intuitive understanding of them. But, in general, it seems possible to define more suitable language primitives by customizing the given $(ML)^2$ language.

In contrast to the other languages, $(ML)^2$ does not distinguish between meta classes and meta relations. $(ML)^2$ restricts the number of output roles to one which forces the modeller to introduce additional pias. If a set should be split into two parts this must be modelled by two pias for example. Hierarchical refinement which helps to structure complex inference structures is not discussed.

Whereas in KARL, for example, only one state of the problem-solving process is stored in the meta classes, in $(ML)^2$ the whole trace of the problem-solving process is stored in the history variables V_{pia} corresponding to every pia. For complex reasoning tasks this will causes problems in operationalization.

It is not possible to ***delete*** contents of knowledge roles, i.e. history variables. The "***old-solution***-$pia_{diagnose}(I_1, I_2, I_3, O)$", for example, asks for all already computed solutions. This restriction is related to the tuple-oriented evaluation strategy of $(ML)^2$. The previous computed solutions of an inference action have to be kept in mind to prevent $(ML)^2$ from always computing the same tuple as its solution.

In general, $(ML)^2$ is designed mainly as a formalization language allowing the declarative description of models of expertise. But experiences show that even for experts in logical formalisms it becomes difficult to really understand the meaning of formally specified knowledge, especially if the model becomes very large. The operationalization of a subset of the language cannot totally overcome this shortcoming, especially because due to differences in the semantics of the formal and the operational language.

4. How can these different languages be compared?

In the last years, a lot of work has been done to overcome the disadvantages of knowledge level specifications only having an informal semantics. The increasing number of specification languages which allow a formal and executable specification of the modelled expertise at the knowledge level try to overcome the shortcomings of life cycle oriented development processes. Informally specified documents like the KADS model of expertise acquire a defined semantics. Three classes of languages can be identified:

- Languages which can be used to operationalize models of expertise, like MODEL-K or OMOS.
- Languages which can be used to formalize models of expertise like $(ML)^2$.
- Languages which attempt both, like FORKADS or KARL.

Therefore, a comparison of these languages has to examine several different cases:

- The comparison of languages which are related, like OMOS and MODEL-K or KARL and FORKADS.
- The comparison of languages in different classes. Whereas the first kind of languages differs significantly from the following ones and only a comparison in general terms, like purpose or usability is possible, operational formalization languages like KARL and non-operational formalization languages like $(ML)^2$ have a lot of features in common. In general, the former define a sublanguage of the latter so as to be able to be of executed. Therefore, the comparison of KARL and $(ML)^2$, for example, shows a lot of similarities, especially for the subset which is defined by $Si(ML)^2$.

Comparing formal and operational languages with formal ones compares in general an (effi-

cient) operational subset of a language with richer formalism. Therefore, the comparison of KARL and $(ML)^2$, for example, shows a lot of similarities, especially for the subset which is defined by $Si(ML)^2$.The main differences between KARL and $(ML)^2$ are:

- At the domain layer, $(ML)^2$ offers the expressive power of first-order logic whereas KARL offers horn logic with finite minimal models.
- The strategy used by KARL to evaluate logical expressions is set-oriented, whereas the evaluation strategy of $(ML)^2$ is tuple-oriented. The tuple-oriented evaluation is used to represent control information at the task layer by $(ML)^2$.
- The inference layer is a real meta layer in $(ML)^2$. Therefore, in principle, it can be used to model inferences about arbitrary domain terms. In KARL, the domain and inference layers are modelled by the same logical language.
- $(ML)^2$ stores the whole problem-solving trace in order to allow to reason about different problem-solving paths. KARL stores only one state of the problem-solving path.

The survey given below provides a concise description of the main characteristics of the three languages having been concerned in this paper:

Survey

	OMOS	KARL	$(ML)^2$
key idea	How to make expert systems shells more flexible?	What can we learn from software engineering and information system design?	How to use different types of logic to formalize models?
(1). Domain layer	- concept hierarchies - relationships - concept instances - extensional and intensional descriptions of relationships	- concept hierarchies - relationship hierarchies - extensional and intensional descriptions of concepts and relationships - constraints	- first-order logic - order-sorted - modularisation
(2). Inference layer	- inference actions (knowledge sources) - roles (control, input, output)	- elementary knowledge sources - composed knowledge sources (hierarchical refinement) - Horn logic to describe knowledge sources - meta classes and meta relations	- primitive inference actions - first-order logic to describe them - roles
(3). Task layer	- sequence, branch, and loop	- sequence, branch, and loop - hierarchical refinement (subtasks) - boolean variables	- ***has-solution***-$pia_i()$ - ***old-solution***-$pia_i()$ - ***more-solution***-$pia_i()$ - ***give-solution***-$pia_i()$ - sequence, loop, and branch of "***give-solution***-$pia_i()$" - variables storing all intermediate results of pias

Survey

	OMOS	KARL	$(ML)^2$
mapping (1)-(2)	- a role can be connected with one domain concept - an inference action can be connected with one domain relation	arbitrary view definitions by function free horn logic extended using stratified negation.	- a lift definition maps domain terms on expressions at the inference layer - reflection rules connect derivations at the meta layer with axioms of the domain layer
mapping (2)-(3)	inference actions and roles return boolean values	inference actions return boolean values	primitive inference actions have a truth value and return variable bindings
speciality	Allows quick prototyping by restricted modelling primitives.	Allows hierarchical structuring of complex models of expertise and has a formal semantics for the declarative parts of an expertise. It is an operationalization and formalization language.	- Realizing inference and task layers as meta layers which allows the reasoning about arbitrary domain terms. - Offers logic as a representation formalism for control knowledge at the task and strategic layer.

Conclusion

The most languages which can be used to formalize or operationalize models of expertise are more complementary than contradictory. In the first step a language like $(ML)^2$ can be used to formalize a model of expertise. For the translation of an informally specified expertise into a formal representation, it is helpful to have as much expressive power as possible. A language like KARL can be used in a second step to add the flavour of executability to the specified expertise. The price for this is a restriction of the expressive power of the language. A language like MODEL-K can help to add the aspect of efficiency to the specified expertise by implementing the bodies of the knowledge sources. This would help support the transition to design and implementation without destroying the conceptual structure of the modelled expertise. A significant deficit in all of the all languages is a lack of support for using tools to assess and correct errors in the modelled expertise. TheME (cf.[THR91]), for example, is a tool which supports the formalization of a model of expertise using $(ML)^2$. But it does not support the evaluation of the modelled knowledge. Only OMOS offers some elementary tools supporting the check for completeness of domain knowledge according to its application at the inference layer. Thus a lot of work still remains to be done.

Acknowledgement

We thank Jürgen Angele, Dieter Landes, Frank van Harmelen, Werner Karbach, Mark Linster, and Angi Voß for helpful and fruitful discussions.

References

[AFL91a] J. Angele, D. Fensel, D. Landes, and R. Studer: Sisyphus - no problem with KARL. In *Proceedings of the 5th European Knowledge Acquisition Workshop EKAW'91*, Crieff, Scotland, May 20-24, 1991.

[AFL91b] J. Angele, D. Fensel, D. Landes, and R. Studer: KARL: An executable Language for the Conceptual Model. In *Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop KAW'91*, Banff, Canada, October 6-11, 1991.

[AFL92a] J. Angele, D. Fensel, and D. Landes: Two Languages to Do the Same? In *Proceedings of the 2nd Workshop Informationssysteme und Künstliche Intelligenz*, February 24-26, Ulm, 1992, Informatik Fachbericht, no 303, Springer Verlag, Berlin, 1992, pp. 23-39.

[AFL92b] J. Angele, D. Fensel, and D. Landes: An executable model at the knowledge level for the office-assignment task. In M. Linster (Hrsg.), Sisyphus '92: Models of problem solving, Arbeitspapiere der GMD, Nr. 630, March 1992.

[AFS90] J. Angele, D. Fensel, and R. Studer: Applying software engineering methods and techniques to knowledge engineering. In D. Ehrenberg, D. et.al. (eds.), *Wissensbasierte Systeme in der Betriebswirtschaft, Reihe betriebliche Informations- und Kommunikationssysteme, no. 15*, Erich Schmidt Verlag, Berlin, 1990.

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

[Alf90] M. Alford: SREM at the age of eight; the distributed computing design system. In R. H. Thayer et.al. (eds.), *System and Software Requirements Engineering*, IEEE Computer Society Press, Washington, 1990, 392-402.

[Ang92] J. Angele: Cover and Differentiate remodeled in KARL. In *Proceedings of the 2nd KADS User Meeting*, München, February 17-18th, 1992.

[AnK93] J. Angele and : 1993.

[BaA92] J. Balder and H. Akkermans: TheMe: an Environment for building Formal KADS-II Models of Expertise, ESPRIT Project P5248 KADS-II, report KADS-II/T1.2/TR/ECN/007/1.0, Netherlands Energy Research Foundation ECN, January 1992.

[BaS89] J. Bachant, E. Soloway E: The Engineering of XCON. In *Communications of the ACM*, vol. 32, no. 3, March, 1989, pp. 311-318.

[Bee90] C. Beeri: A formal approach to object-oriented databases. In *Data & Knowledge Engineering*, vol. 5, 1990, pp. 353-382.

[Ben56] H.D. Benington: Production of Large Computer Programs. In *Proceedings ONR symposium on Advanced Programming Methods for Digital Computers*, June 1956, pp. 15-27.

[Ber87] D. C. Berry: The problem of implicite knowledge, *Expert Systems*, vol 4, no 3, August 1987, pp. 144-151.

[Boe86] B.W. Boehm: A Spiral Model of Software Development and Enhancement. In *ACM SIGSOFT Software Engineering Notes*, vol. 11, no. 4, 8/1986, pp. 14-24.

[Boe88] B.W. Boehm: A Spiral Model of Software Development and Enhancement. In: *Computer*, May 1988.

[Bra85] R. J. Brachman: On the Epistemological Status of Semantic Networks. In Readings in Knowledge Representation, 1985, pp. 191-215.

[Bra89] I. Bratko: Fast prototyping of expert systems using Prolog. In G. Guida et.al. (eds.), *Topics in Expert Systems Design*, Elsevier Science Publisher B. V., North-Holland, 1989.

[BrR84] M.L. Brodie and D. Ridjanovic: On the design and specification of database transactions. In M.L. Brodie et.al. (eds.), *On Conceptual modelling. Perspectives from Artificial Intelligence, Databases, and Programming Languages*, Springer-Verlag, New York, 1984.

[BrW89] J. Breuker and B. Wielinga: Models of Expertise in Knowledge Acquisition. In G. Guida et.al. (eds.), *Topics in Expert Systems Design*, Elsevier Science Publisher B. V., North-Holland, 1989.

[BRW90] B. Bredeweg, M. Reinders, and B. Wielinga: Garp: A unified approach to qualitative reasoning, report VF-memo 117, University of Amsterdam, 1990.

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

[Che76] P.P. Chen: The entity relationship model. Toward a unified view of data. *ACM Transactions on Database Systems* 1, 1, 1976, 9-36.

[Cla85] W.J. Clancey: Heuristic Classification. In *Artificial Intelligence*, vol. 27, 1985, pp. 289-350.

[Cla89] W.J. Clancey: The Knowledge Level Reinterpreted: Modeling How Systems Interact. In: *Machine Learning*, vol 4, 1989, 285-291.

[Cla92] W.J. Clancey: The Knowledge Level Reinterpreted: Modeling in Socio-Technical Systems. To appear in: *The International Journal of Intelligent Systems*

[CPV89] T. Christaller, F. di Primo, and A. Voß: *Die KI-Werkbank BABYLON*, Addison Wesley, Bonn, 1989.

[CSB82] D. Cohen, W. Swartout, and R. Balzer: Using Symbolic Execution to Characterize Behavior. In *ACM Sigsoft Software Engineering Notes*, vol. 7, no. 5, 1982, pp. 25-32.

[Dub89] E. Dubois: A logic of action for supporting goal-oriented elaborations of requirements. In 5th International Workshop on Software Specification & Design, 1989, 160-168.

[FAL91] D. Fensel, J. Angele, and D. Landes: KARL: A Knowledge Acquisition and Representation Language. In *Proceedings of Expert Systems and their Applications, 11th International Workshop, Conference "Tools, Techniques & Methods"*, 27-31 May, Avignon, 1991, pp. 513-528.

[Flo84] Floyd, C.: A systematic look at prototyping. In R. Budee et. al. (eds.), *Approaches to Prototyping*, Springer-Verlag, Berlin, 1984.

[FRS89] C. Floyd, F.-M. Reisin, and G. Schmidt: STEPS to Software Development With User. In Proceedings of the 2nd European Software Engineering Conference ESEC´89, Warwick, Lecture notes in Computer Science, no. 387, Springer Verlag, 1989.

[GMD91] German National Resaerch Center for Computer Science (GMD): Reflective modules on top of OFFICE-PLAN. In II.2 report, ESPRIT-Basic Research Project P3178, Juni 1991.

[HaB92] F. v. Harmelen and J. Balder: $(ML)^2$: A formal language for KADS conceptual models. In *Knowledge Acquisition*, vol. 4, no. 1, March 1992, pp. 127-161.

[HWL83] F. Hayes-Roth, D. Waterman, and D. Lenat (eds.): *Building Expert systems*, Addison-Wesley, Reading, London, 1983.

[JoS92] W. Jonker and J.W. Spee: Yet another formalisation of KADS conceptual models. In *Proceedings of the 6th European Knowledge Acquisition for Knowledge-Based Systems Workshop (EKAW-92)*, May 18-22, Heidelberg/Kaiserslautern, 1992.

[KaV92a] W. Karbach and A. Voß: An inference structure for assignment problems. In *Proceedings of the 2nd KADS User Meeting*, München, February 17-18th, 1992.

[KaV92b] W. Karbach and A. Voß: Reflecting about expert systems in MODEL-K. In *Proceedings of Expert Systems and their Applications, 12th International Workshop, vol 1 (Scientific Conference)*, June 1-6, Avignon, 1992, pp. 141-152.

[KaV92c] W. Karbach and A. Voß: MODEL-K for prototyping and strategic reasoning at the knowledge level. In J.M. David, J.S. Krivine, and R. Simmons (eds.), *Second Generation Expert Systems*, forthcoming.

[KFG92] R. Köppen, D. Fensel, J. Geidel: Modelling the Selection of Scheduling Algorithm with KARL. In *Proceedings of the 2nd KADS User Meeting*, München, February 17-18th, 1992.

[KLS91] O. Kühn, M. Linster, and G. Schmidt: Clamping, COKAM, KADS, and OMOS: The Construction and Operationalization of a KADS Conceptual Model. In *Proceedings of the 5th European Knowledge Acquisition Workshop EKAW'91*, Crieff, Scotland, May 20-24, 1991.

[Kun89] C. H. Kung: Conceptual Modeling in the Context of Software Development. In *IEEE Transaction on Software Engineering*, vol. 15, no. 10, 1989, 1176-1187.

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

[LFA92] D. Landes, D. Fensel, and J. Angele: Formalizing and Operationalizing a Design Task with KARL, *Proceedings of the ECAI-92 Workshop Formal Specification Methods for Complex Reasoning Systems, European Conference on Artificial Intelligence, Wien, Austria*, , 1992.

[LHS92] D. Landes, D. Hackenberg, and T. Schweier: An Inference Structure for a Configuration Problem. In *Proceedings of the 2nd KADS User Meeting*, München, February 17-18th, 1992.

[LiM92] M. Linster and M. Musen: The Inference Structure of K-ONCOCIN: Skeletal Plan Refinement. In *Proceedings of the 2nd KADS User Meeting*, München, February 17-18th, 1992.

[Lin91] M. Linster: Sisyphus`91 Part 2: Tackling the Office-Plan Problem with OMOS. In *Proceedings of the 5th European Knowledge Acquisition Workshop EKAW'91*, Crieff, Scotland, May 20-24, 1991.

[Lin92a] M. Linster: Knowledge Acquisition Based on Explicit Methods of Problem Solving, PhD thesis, University of Kaiserslautern, February 1992.

[Lin92b] M. Linster: Linking modeling to make sense and modeling to implement systems in an operational modeling environment. In *Proceedings of the 6th European Knowledge Acquisition for Knowledge-Based Systems Workshop (EKAW-92)*, May 18-22, Heidelberg/Kaiserslautern, 1992.

[Llo87] J.W. Lloyd: *Foundations of Logic Programming*, second edition, Springer Verlag, Berlin, 1987.

[LNO89] G. Lausen, T. Nemeth, A. Oberweis, F. Schönthaler, and W. Stucky: The INCOME approach for conceptual modelling and prototyping of information systems. In *Proceedings of the 1st Nordic Conference on Advanced Systems Engineering CASE'89*, Stockholm, Sweden, May 9-11, 1989.

[Mar88] S. Marcus (ed.): *Automating Knowledge Acquisition for Experts Systems*, Kluwer Academic Publisher, Boston, 1988.

[Mus89a] M. Musen: Conceptual models of interactive knowledge acquisition tools. In *Knowledge Acquisition*, vol. 1, nr. 1, March 1989, pp. 73-98.

[Mus89b] M. Musen: *Automated Generation of Model-Based Knowledge-Acquisition Tools*, Morgan Kaufmann Publisher, San Mateo, CA, 1989.

[Mor87] K. Morik: Sloppy modeling. In K. Morik (ed.), *Knowledge Representation and Organisation in Machine Learning*, Lecture Notes in Artificial Intelligence, no. 347, Springer-Verlag, Berlin, 1987.

[Mor91] K. Morik: Underlying assumptions of knowledge acquisition and machine learning. In *Knowledge Acquisition*, vol. 3, 1991, 137-156.

[NeS92] S. Neubert and R. Studer: The KEEP model. In *Proceedings of the 6th European Knowledge Acquisition for Knowledge-Based Systems Workshop (EKAW-92)*, May 18-22, Heidelberg/Kaiserslautern, 1992.

[New82] A. Newell: The knowledge level, *Artificial Intelligence*, vol. 18, 1982, pp. 87-127.

[Rei90] W. Reisig: *Petrinetze. Eine Einführung*, second edition, Springer-Verlag, Berlin, 1990.

[Ros77] D.T. Ross: Structured Analysis (SA): a language for communicating ideas. *IEEE Transactions on Software Engineering SE-3*, 1 (January 1977), pp. 16- 34.

[Roy70] W. W: Royce: Managing the Development of Large Software Systems: Concepts and Techniques. In *Proceedings WESCON*, August 1970

[SAW89] G. Schreiber, H. Akkermans, and B. Wielinga: On Problems with the Knowledge Level Perspective. In *Proceedings of the 5th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, November, 1989.

[Say90] H. H. Sayani: PSL/PSA at the Age of Fifteen. In R. H. Thayer et.al. (eds.), *System and Software Requirements Engineering*, IEEE Computer Society Press, Washington, 1990, 403-417.

[SBJ87] E. Soloway, J. Bachant, K. Jensen, K.: Assessing the Maintainability of XCON-in-RIME: Coping with the Problems of a VERY large Rule-Base. In *Proceedings of 6th National Conference on AI (NCAI)*, AAAI, Seattle, Washington, July 13-17, 1987, pp. 824-829.

[SWA92] G. Schreiber, B. Wielinga, and H. Akkermans: Differentiating Problem Solving Methods, . In *Proceedings of the 6th European Knowledge Acquisition for Knowledge-Based Systems Workshop (EKAW-92)*, May 18-22, Heidelberg/Kaiserslautern, 1992.

[THR91] A. t. Teije, F.v. Harmelen, and M. Reinders: Si(ML)^2: a prototype interpreter for a subset of (ML)^2, ESPRIT project P5248 KADS-II, report KADS-II/T1.2/TR/UvA/005/1.0, University of Amsterdam, 1991.

[Ull88] J.D. Ullman: *Database and Knowledge-Base Systems, vol. 1*, Computer Science Press, New York, 1988.

[VJS92] L. in 't Veld, Willem Jonker, and J. W. Spee: The Specification of Complex Reasoning Tasks in $K_{BS}SF$, *Proceedings of the ECAI-92 Workshop Formal Specification Methods for Complex Reasoning Systems, European Conference on Artificial Intelligence (ECAI-92), Wien, Austria*, August 3-7, 1992, pp. 207-230.

[VKS91] A. Voß, W. Karbach, R. Schuckey, and U. Drouven: The Office Planning Problem in MODEL-K: In *Proceedings of the 5th European Knowledge Acquisition Workshop EKAW'91*, Crieff, Scotland, May 20-24, 1991.

[Wet90] T. Wetter: First order logic foundation of the KADS conceptual model. In *Current Trends in Knowledge Acquisition*, B. Wielinga et. al. (eds.), IOS Press, Amsterdam, 1990.

[WeS91] T. Wetter and W. Schmidt: Formalization of the KADS Interpretation Models. In Proceedings of the 8th Conference of the Society for the Study of Artificial Intelligence and Simulation of Behaviour (AISB91), Leeds, April 16-19, 1991.

[WSB92] B.J. Wielinga, A.Th. Schreiber, and J.A. Breuker: KADS: A Modelling Approach to Knowledge Engineering. In *Knowledge Acquisition*, vol. 4, no. 1, March 1992, pp. 127-161.

[You89] E. Yourdon: *Modern Structured Analysis*. Prentice Hall, Englewood Cliffs, 1989.

[Zav91] P. Zave: An insider's evaluation of PAISLey. In *IEEE Transactions on Software Engineering*, vol. 17, no. 3, 1991.

KWRITE: INCORPORATING ACTIVE KNOWLEDGE BASES IN MULTI-MEDIA DOCUMENTS

Brian R Gaines & Mildred L G Shaw

Knowledge Science Institute, University of Calgary
Calgary, Alberta, Canada T2N 1N4.
gaines@cpsc.ucalgary.ca & mildred@cpsc.ucalgary.ca

Abstract: This paper is written in a document production tool that appears to a user as a word processor but also acts as an expert system shell with frame and rule representations supporting deductive inference. The electronic version of the document is active, providing typographic text and page layout facilities, versioning, hypermedia sound and movies, hypertext links, and knowledge structures represented in a visual language. It can be read as a hypermedia document and also interrogated as a knowledge-based system for problem-solving. The paper version of the document, which you are now reading, is produced by printing the electronic version. It loses its active functionality but continues to act as a record of the knowledge in the document. The overall technology has been developed as an alternative approach to the dissemination of knowledge bases. It also provides a different interface to knowledge-based systems that emulates document interfaces with which many users are already familiar.

1 INTRODUCTION

The knowledge document publication system emulates conventional word-processing packages as closely as possible to require the minimum of new skills in the user. It produces documents that are formatted and paginated for printing so that parallel publication of paper and electronic documents is available. It allows diagrams, pictures, video and sound to be integrated in documents, with their preparation and editing being based on existing packages so that again the user has the minimal learning requirements. For example, the picture inset in this paragraph is a QuickTime video with moving picture and sound commentary that may be played by double clicking in the picture. The mechanism for linking to multi-media material is also used to provide simple and versatile hypertextual linking. Formal knowledge structures may also be embedded in documents, represented as semantic networks in a visual language easily understood by people. The same knowledge structures may also be accessed through computer programs to provide the decision support and problem solving capabilities of an expert system.



The knowledge document publication system is part of a research program on general 'knowledge support systems' integrating many information technologies to support knowledge processes in human society (Gaines, 1990). Since the intended users are not computer specialists, our objective has been to achieve all the functionality outlined above through an environment that appears simple and natural to the user. That is, the project is successful to the extent that the system appears like a conventional word processor rather than an over-engineered concatenation of multi-media and artificial intelligence functionality. This paper focuses on the expert system functionality of the system and illustrates this through some studies of organizational modeling and problem solving which were reported in greater detail in an (inactive) paper at the Conference on Organizational Computing Systems (Gaines, 1991d).

The problem considered is one of room allocation from an ESPRIT project (Voß, Karbach, Drouven, Lorek & Schuckey, 1990) that has recently been made part of Project Sisyphus. Sisyphus is a research program to encourage international collaboration in knowledge-based system development initiated by the European Knowledge Acquisition Workshop in 1989. A number of problem datasets have been made available through Sisyphus, and a major part of the EKAW'91 program was devoted to reports on the solution of these problems using different approaches and techniques (Linster, 1991).

What is remarkable about the document you are reading is that the paper version of it reports a solution to the problem, giving all the knowledge structures involved in visual form, and the electronic version of it *is* the solution. That is, if you opened this paper in the associated word processor you could interrogate it to solve room allocation problems. You could edit the visual knowledge structures within the document, for example by adding additional rules, and when you interrogated it again those rules would be in effect. Thus, the document itself provides an active, editable knowledge base and problem-solving inference engine.

The next section gives an overview of the architecture of the system, followed by the visual language for knowledge representation and the problem formulation and solution in this language.

2 KSSn ARCHITECTURE

KSSn (Knowledge Support System n) is the latest in a series of developments deriving from our initial implementation of KSS0 (Gaines, 1988a,b) and KSS1 (Shaw & Gaines, 1987). These early knowledge support systems focused on knowledge acquisition and conceptual modeling, and have been extended through heterogeneous integration to offer close integration with hypermedia and expert system shells (Gaines, Rappaport & Shaw, 1989; Gaines & Linster, 1990). KSSn is designed as a C++ class library implementing KRS (Gaines, 1991a), a KL-ONE-like (Brachman & Schmolze, 1985) knowledge representation server, and a set of associated functional modules for knowledge elicitation, text analysis, empirical induction, graphic knowledge base editing, and so on.

Figure 1 shows the architecture of KSSn as a family of modules attached to the knowledge representation server, KRS. The modules are (clockwise from the top left):

- Interface modules to other knowledge bases and servers, including databases.
- A hypermedia module allowing informal knowledge structures in text and images to be captured, accessed and linked. The linkage structure is held as a knowledge base.
- A text analysis module allowing documents to be analyzed in terms of word usage, and associations between significant words to be graphed—based on TEXAN in KSS0. This enables protocols and technical documents to be used to initiate knowledge acquisition.
- A repertory grid expertise transfer module allowing graphic definition of concepts and graphic creation and editing of individuals—based on the elicitation screens of KSS0.
- A conceptual clustering module allowing interactive definition of new concepts—based on the hierarchical and spatial clustering from KSS0.
- A knowledge editing module allowing the interactive development and editing of knowledge structures through a visual language.
- A conceptual induction module creating rules about specified subsets of individuals and transforming them to a minimal set of concepts and default rules—based on the Induct algorithm.
- A problem solving module supporting frame, rule and case-based inference from the knowledge structures.
- A grapher laying out specified parts of the concept subsumption graph, concept structures and individual structures—using an incremental layout algorithm that can be used interactively to support the production of clear visual knowledge structures.
- A language interface accepting and generating definitions and assertions in formal knowledge representation languages, both textual and visual.

The knowledge representation services of KRS, the central server module, correspond to those of CLASSIC (Borgida, et al, 1989), augmented with inverse roles, data types for integers, reals, strings and dates, and with rule representation that allows one rule to be declared an exception to others. KWrite, the document production tool used to produce this paper, may be seen as providing a word processing user interface to the functionality shown in Figure 1.

3 KDRAW VISUAL LANGUAGE

An important component of KWrite and KSSn in the context of this paper is the graphic knowledge editor, KDraw, at the right of Figure 1. This is a drawing tool designed for ease of use that provides a visual structure editor for semantic networks representing classes, objects and rules in KRS. Nosek and Roth (1990) have demonstrated empirically that the visual presentation of knowledge structures as semantic nets leads to more effective human understanding than does textual presentation of the same structures. We have developed a formal visual language that corresponds exactly to the underlying algebraic semantics of KRS that has remarkably few visual primitives and is easily learnt and understood (Gaines & Shaw, 1990).

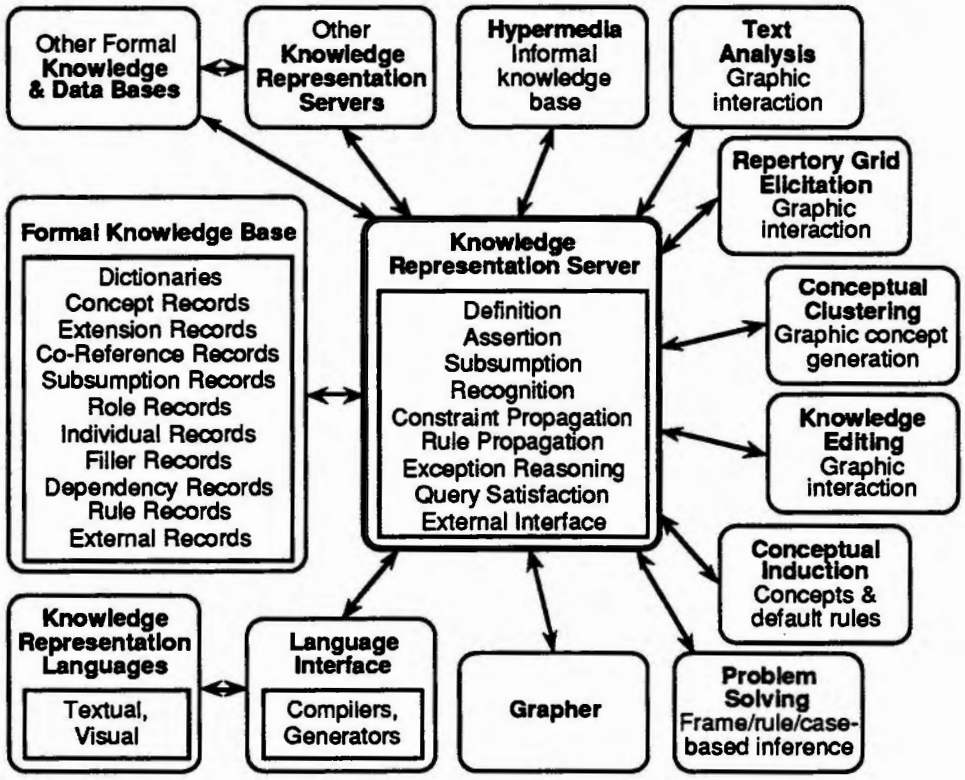


Figure 1 Architecture of the knowledge representation server

Visual representation of knowledge structures has been common since the early development of diagrams and taxonomies, and was associated with semantic networks in the early days of artificial intelligence (Quillian, 1968). The early development of such nets resulted in criticisms that the semantics of particular diagrams was not well-defined (Woods, 1975; Brachman, 1977). Nodes, arcs and their labels could be used very freely and ambiguously and diagrams were subject to differing interpretations. In the 1970s there were proposals for network formalisms with well-defined semantics (Cercone & Schubert, 1975; Fahlman, 1979; Brachman, 1979). However, these preceded two important developments in computing: first, the ubiquity of personal workstations with high resolution graphics supporting visual languages as operational editors (Glinert, 1990); second, the studies of complexity issues in knowledge representation, leading to the simplified and tractable semantics of CLASSIC (Borgida et al, 1989).

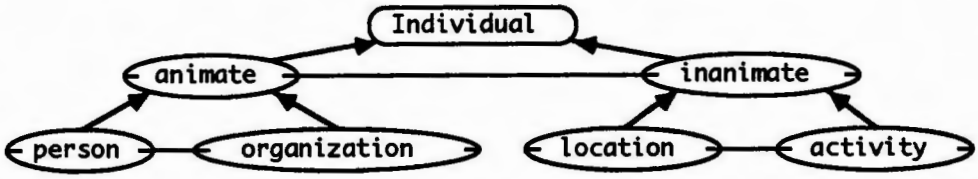
Computer production of visual forms of knowledge represented in a computer has been a topic of research since the early days of knowledge representation research (Schmolze, 1983) and a feature of many research systems (Kindermann & Quantz, 1989) and commercial products. Abrett and Burstein's (1988) KREME system graphically displays the computed subsumption relations between concepts so that those entering knowledge structures can see the consequences of definitions and detect errors due to incorrect or inadequate definitions.

The KDraw design (Gaines, 1991c) has drawn upon this previous research and experience to develop the visual syntax and underlying semantics of a visual language for term subsumption knowledge representation languages in the KL-ONE family. It focuses on the use of the language to enter and edit knowledge visually, and on its application in a highly interactive graphic structure editor. KDraw may be used just to support the entry of conceptual structures and facts in a knowledge base. However, it is capable of going beyond this and accepting problem solving knowledge structures in the form of concepts defining the premise and conclusion of rules. The rule structure provided in KDraw and KRS is itself powerful in both its representational and inference capabilities in supporting exceptions and defaults (Gaines, 1991b).

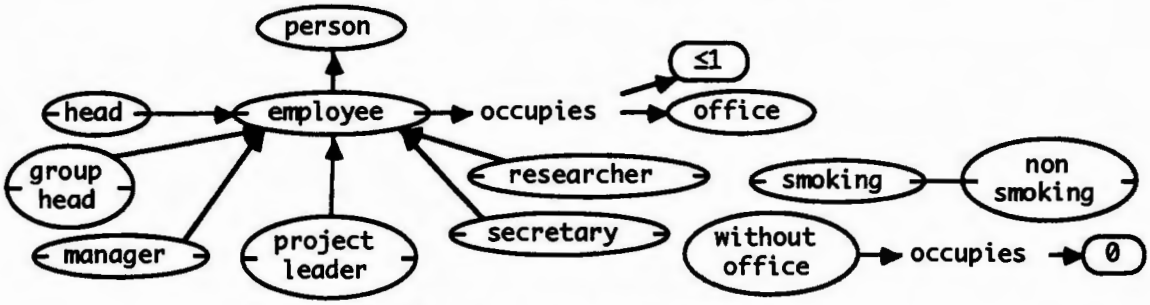
The structures below show the top level conceptual structures of an organizational domain. The visual language used in these diagrams is precisely defined. Concepts are ovals, primitive concepts are ovals with small horizontal lines inside each side, individuals are rectangles, roles are unboxed text, rules are rectangles with double lines at the sides, constraint expressions are rounded-corner boxes. Lines without arrows connecting primitive concepts denote that the concepts are disjoint, and those connecting roles denote that they are inverse. The interpretation of the arrows in the editor is overloaded but well-defined by the types of the objects at their head and tail, e.g.:

concept → concept	definitional subsumption
concept → role → concept	definitional role with conceptual constraint
concept → role → constraint	definitional role with extensional, cardinality or numeric constraint
constraint → individual	extensional constraint
individual → concept	asserted constraint on individual
individual → role → individual	asserted value of role for individual
concept → rule → concept	production rule
rule → rule	first rule exception to second

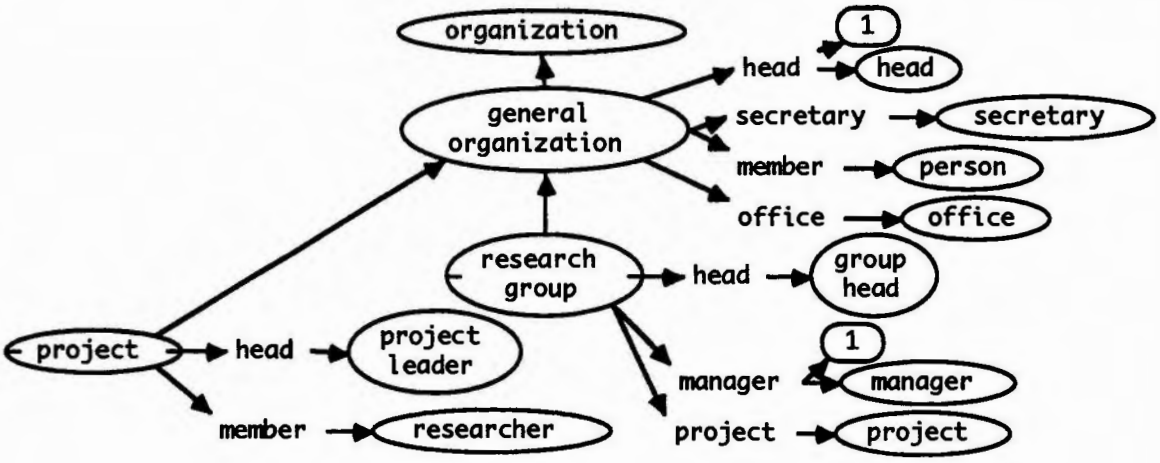
Thus knowledge structure 1 defines "animate" and "inanimate" to be disjoint primitive concepts of type "Individual"; "person" and "organization" to be disjoint concepts inheriting from "animate"; and "location" and "activity" to be disjoint concepts inheriting from "inanimate". Knowledge structure 3 further defines "general organization" as an "organization" (nodes may be freely duplicated for the sake of visual appearance) to have the role "head" filled by exactly one individual of type "head", to have the role "secretary" possibly filled by individuals of type "secretary", and to have the role "member" possibly filled by individuals of type "person". A project, since it is shown to inherit from "organization" also has these roles and constraints but is further constrained to have its "head" role filled by a "group head" and its "member" role filled by individuals of type "researcher".



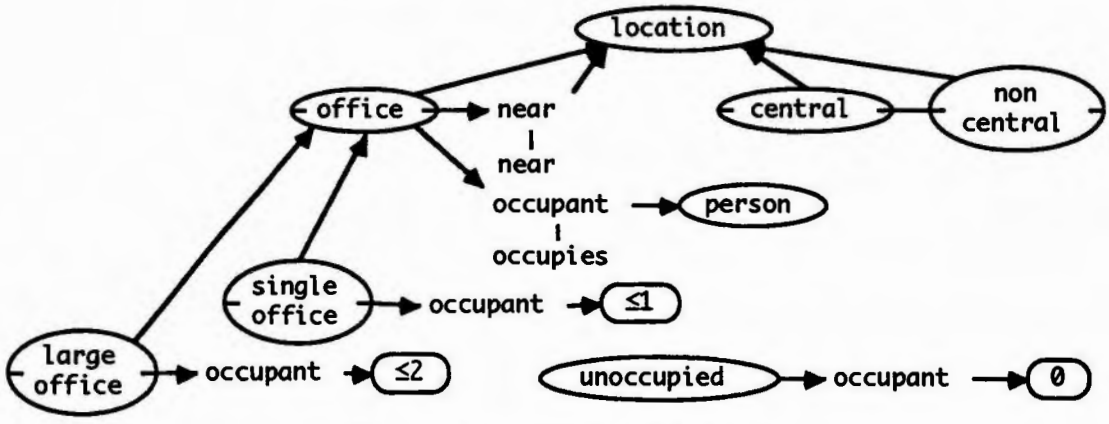
Structure 1: Top Level Ontology



Structure 2: Employee Ontology



Structure 3: Organization Ontology



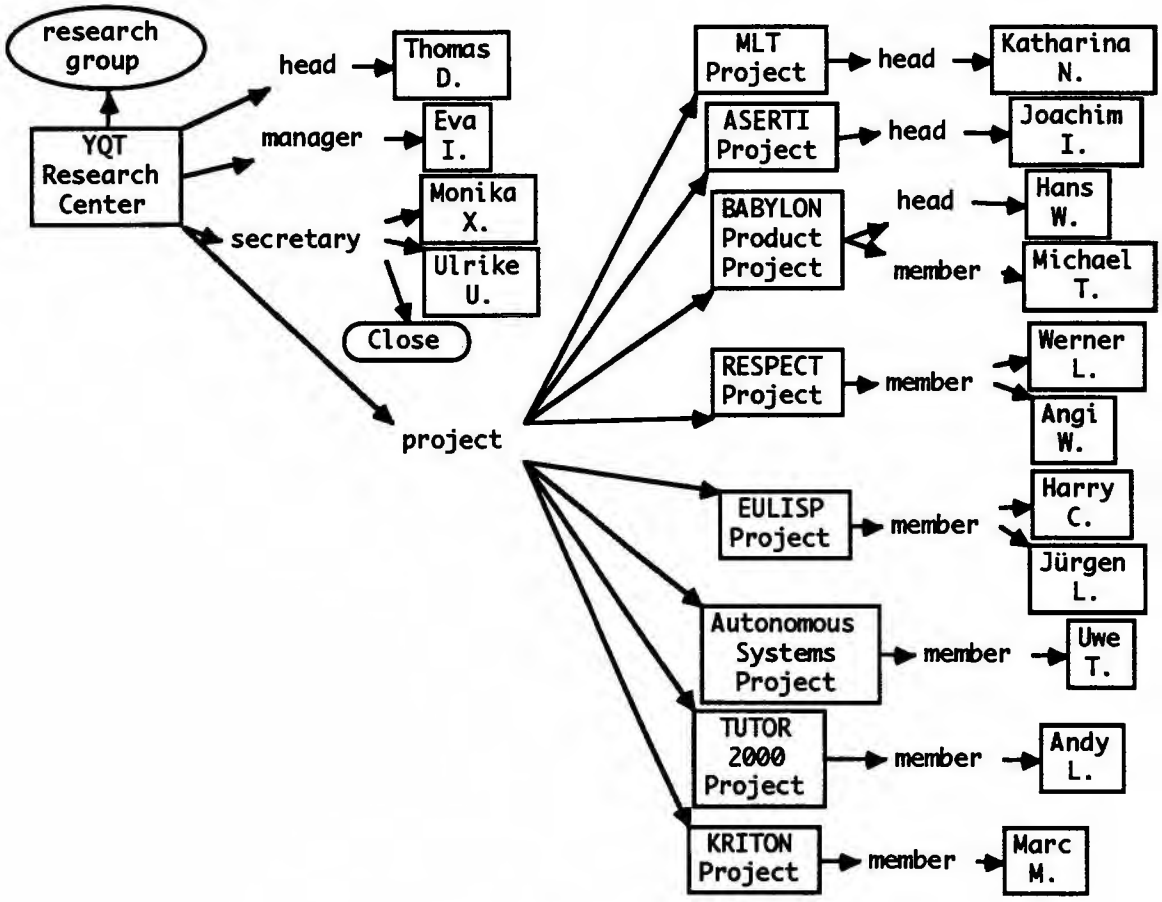
Structure 4: Office Ontology

The underlying knowledge representation inference engine propagates constraints so that defining an individual as a "project" and then filling its "member" role with "Marc M." will automatically lead to "Marc M." being inferred to be a "researcher". Structure 4 shows some other features of the language. For example "occupant" and "occupies" are defined to be inverses, as are "near" and "near". Hence, if "Marc M." is asserted to fill the "occupant" role of office "C5-120" then it will be inferred that "C5-120" fills the "occupies" role of "Marc M.", and if "C5-119" is asserted to fill the "near" role of office "C5-120" then it will be inferred that "C5-120" fills the "near" role of "C5-119."

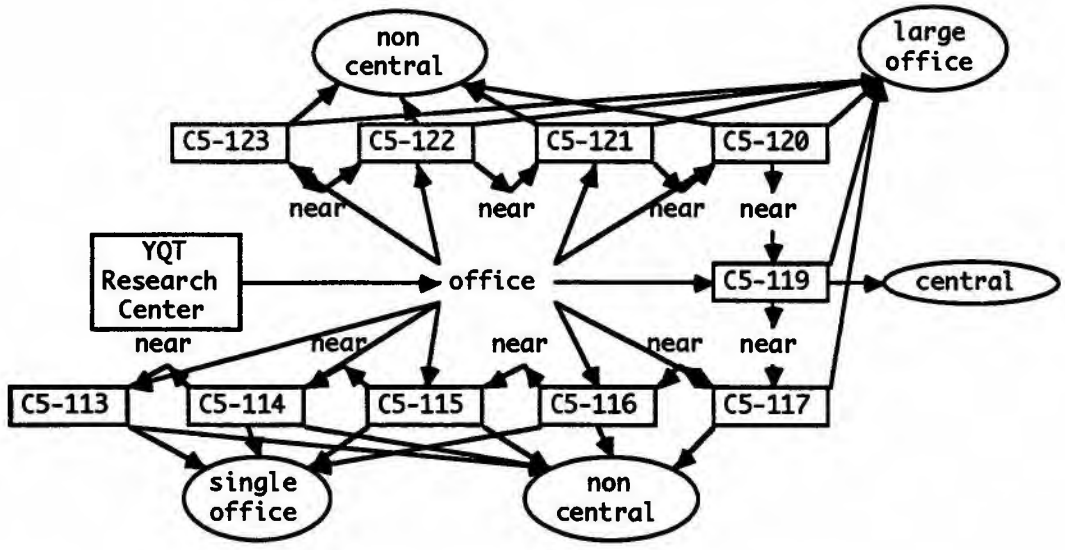
Structures 5 and 6 show the way in which facts may be asserted about individuals in the visual language. For example, the "RESPECT Project" in structure 5 is shown to be one of the fillers of the "project" role of "YQT Research Center" which is defined to be an instance of a "research group" defined in structure 3. This concept instantiation has the consequence that the conceptual constraints defined in structures 1 through 3 will be inherited appropriately by individuals in structure 5. For example, from structure 3 it will be inferred that "Thomas D." is "group head", "head", and the only filler of the "head" role, from structure 2 that he is an "employee" and a "person", and from structure 1 that he is "animate" and of type "Individual". Structure 6 supplies further facts about the project and the rooms that will be needed in the problem-solving activity.

These structures have a number of significant features as semantic networks. First, they are fully operational. They were created in the KDraw graphic structure editor and compile directly into knowledge structures in KRS. In this document they are active as well as operational, and can be edited with immediate impact on inference. Second, the visual language used is completely formally defined and intertranslatable with the underlying KL-ONE knowledge structures. Third, the freedom in layout has been used to create knowledge structures that are natural to the people involved. Structure 5 looks like an organization chart. Structure 6 is based on the actual room layout in the building. Fourth, editing these structures changes the ontologies and facts, and hence any related problem solving activity. For example, at the top right of the room layout, room "C5-119" is the only room asserted to be "central". If this appears to be restrictive when the room allocation rules are run then it is easy for the user to add arrows from "C5-117" and "C5-120" to"central" and see what changes result in the room allocation behavior.

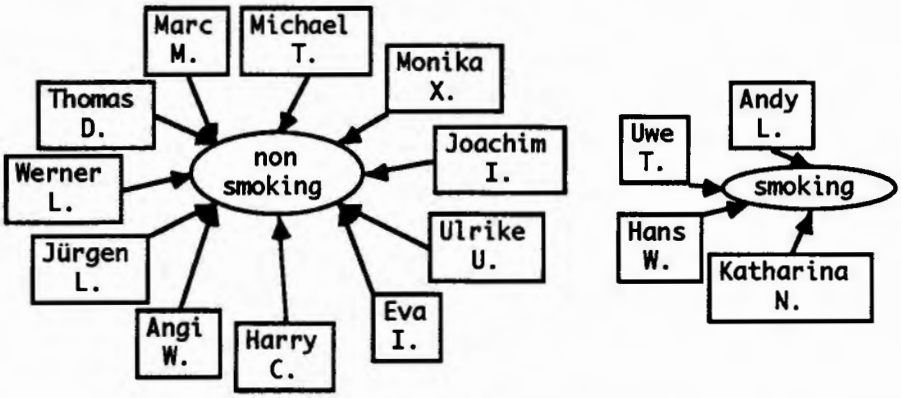
Double-clicking in the structures above brings up a floating dialog box allowing them to be edited as shown in Figure 2. Human-computer interaction in the editor is modeled on Apple's MacDraw with additional features appropriate to the language such as arcs remaining attached to nodes when they are dragged. A popup menu that appears when one mouses down on the right edge of a node allows connecting lines to be entered easily. The syntax of possible node interconnections and constraint expressions is enforced—it is not possible to enter a graph that is syntactically incorrect. Cut-and-paste of graphs and subgraphs is supported, and scroll and fit-to-size capabilities allow large structures with a thousand or more nodes to be edited in KDraw.



Structure 5: Organization Chart



Structure 6: Room Layout



Structure 7: Smoking Employees

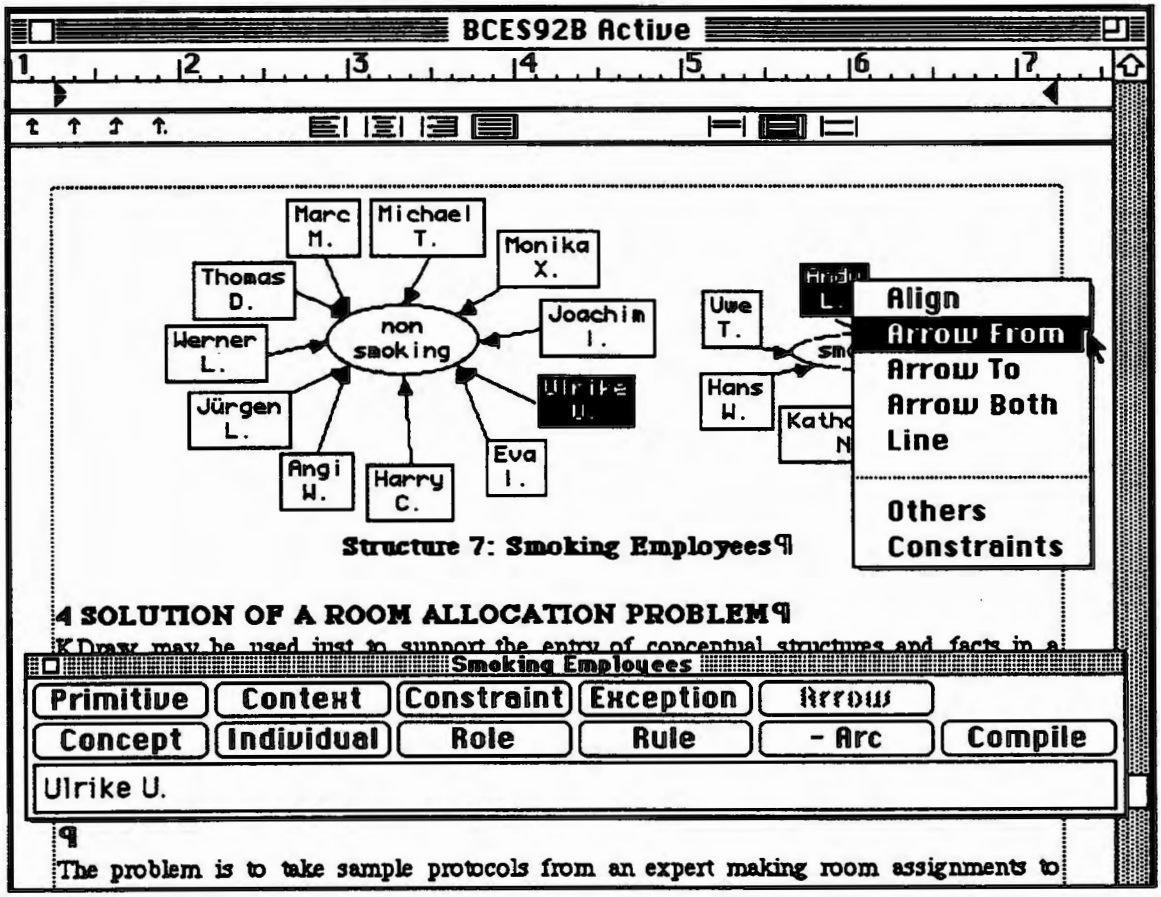


Figure 2 Screen dump of this page of this paper being edited in KWrite. Double clicking on knowledge structure 7 above has brought up a floating dialog box to edit it. A popup menu has been accessed at the right edge of a particular graphic item in order to enter a connecting arrow. Some of the word-processing features of KWrite are also apparent in this figure.

4 SOLUTION OF A ROOM ALLOCATION PROBLEM

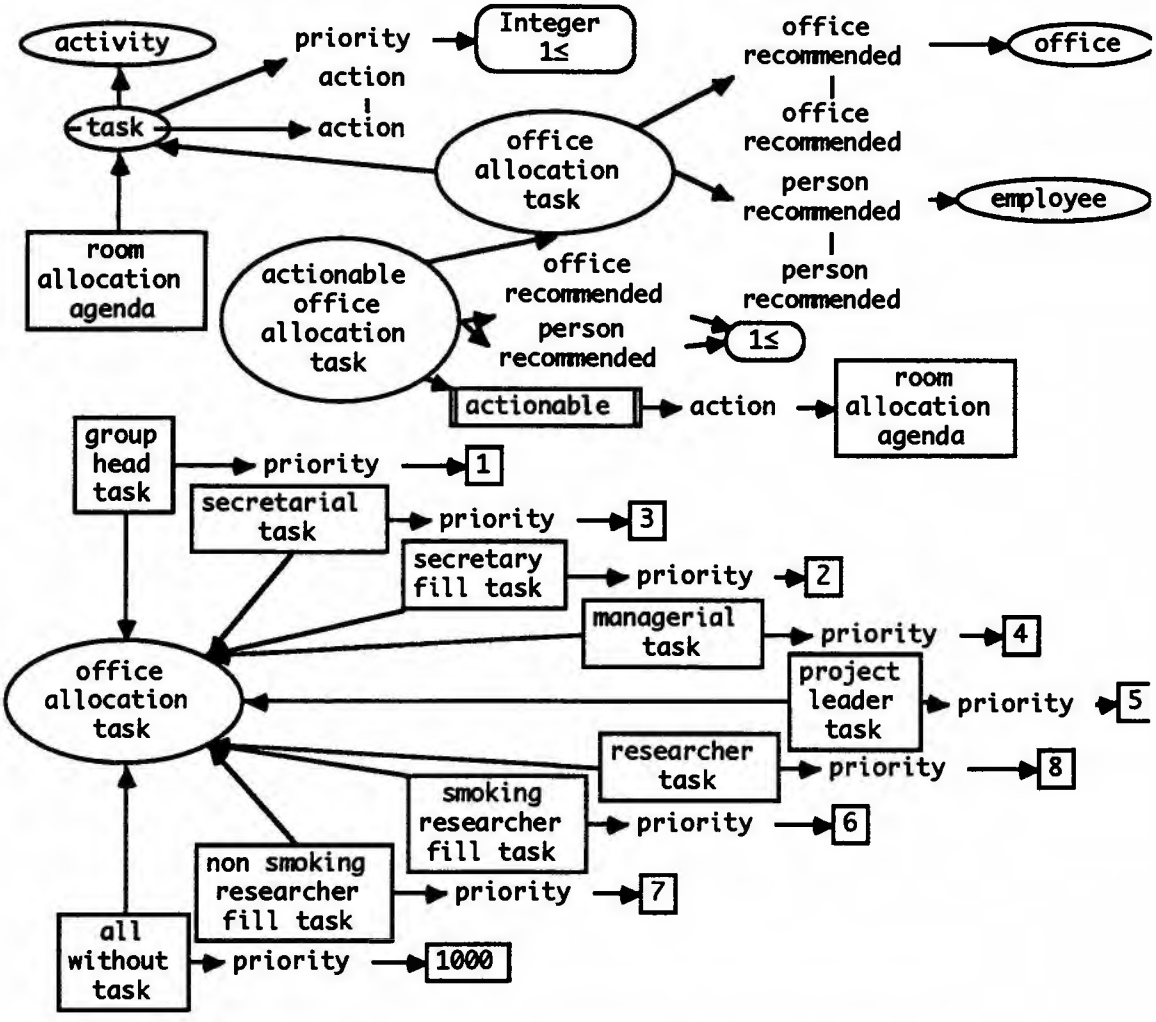
The problem is to take sample protocols from an expert making room assignments to researchers occupying a new building. The rules derived from the expert protocols are, in order of declining priority:

A large, central office should be allocated to the Head of Group.
A large office already occupied by a secretary should be allocated to another secretary.
A large office near the Head of Group is suitable for the secretaries.
A single office near the Head of Group is suitable for the Manager.
A single office near the Head of Group is suitable for a Project Leader.
An office with one smoking researcher occupant should be allocated to another smoking researcher.
An office with a non-smoking researcher occupant should be allocated to another non-smoking researcher.
A large office should be allocated to researchers.

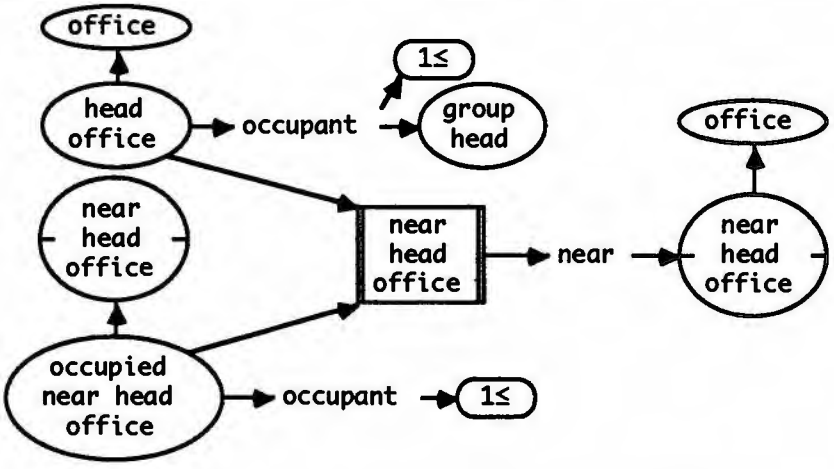
It is not sufficient just to implement these rules. Resource allocation problems tend to be either over-determined, and hence notionally insoluble, or under-determined, and hence subject to combinatorial explosion. People deal with this by problem reformulation, which is a strongly knowledge-based process, and this is expedited by systems that generate meaningful partial solutions that indicate the sources of obstacles to solution. What is required is a system that uses the rules to suggest allocations, allowing the user to make choices when the problem is under-determined, and to resolve conflicts when the problem is over-determined. The system should support retraction and backtracking if the user wishes to explore alternative solutions, perhaps involving considerations not expressed in the knowledge base. An agenda mechanism to support this approach is itself programmed as knowledge structure 8, with the concept of a prioritized task defined at the top and appropriate tasks and priorities defined at the bottom.

Structure 9 shows a rule to determine if a room is "near" to that of the "group head". The upper left concept defines a "head office" as an "office" with at least one occupant and every occupant a "group head". The rule "near head office" asserts that an office "near" such an office is "near head office"—the concept "near head office" will be asserted of it. The lower left concept defines an "occupied near head office" office as one which is "near head office" and has at least one occupant. Such an office again has its neighbors classified as "near head office". This captures the expert's reclassification of offices further away from head office as being "near" to it as intervening offices are filled.

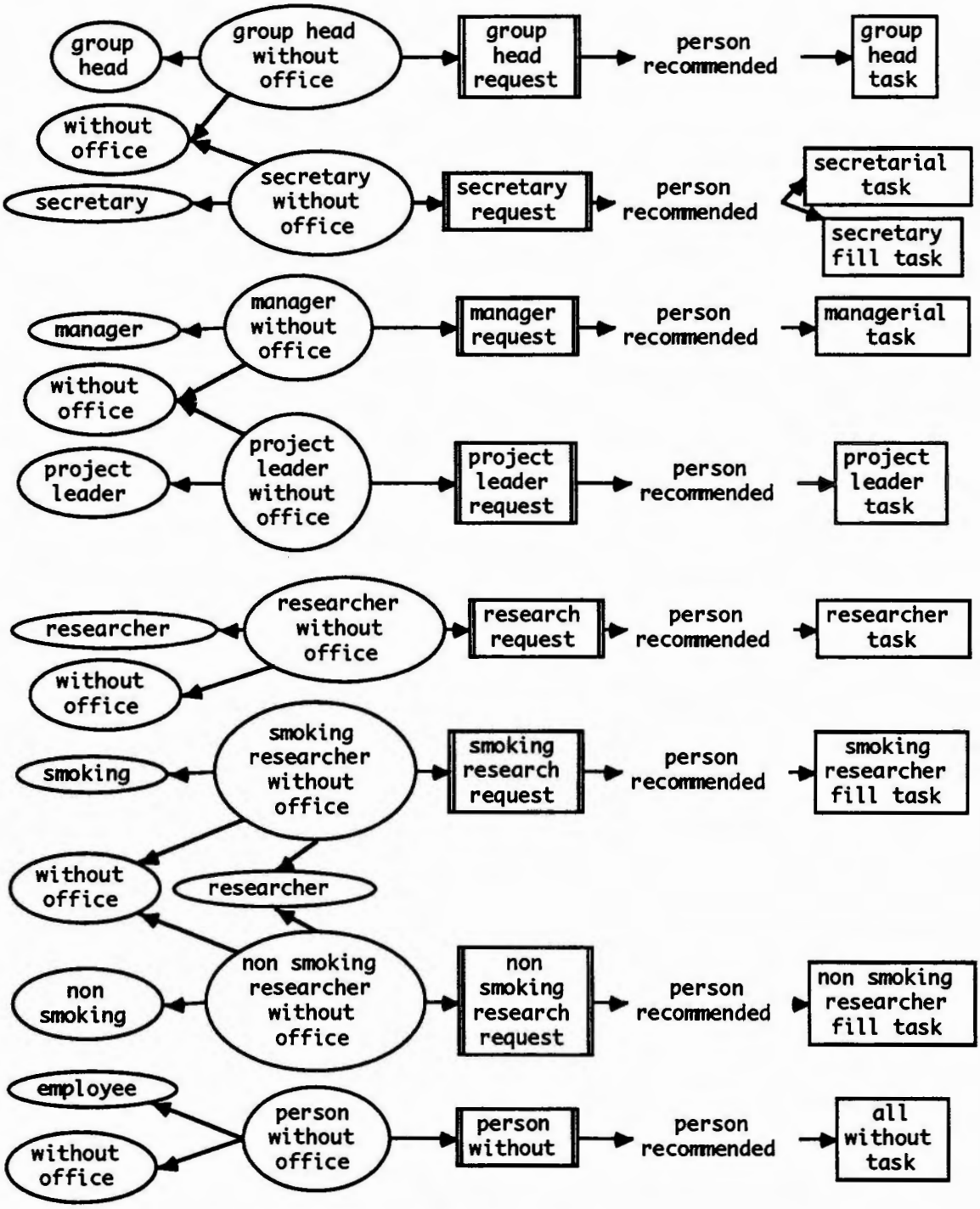
Structures 10 and 11 show the representation of the rules above. For example, at the top of structure 10, if an individual is a "group head" and "without office" (both defined in structure 2) then that individual is classified as "group head without office" and the "group head request" rule fires placing "group head task" in the "person recommended" role of the individual. Hence, since this role is defined as self-inverse in structure 8, the individual is inferred to be in the "person recommended" role of "group head task".



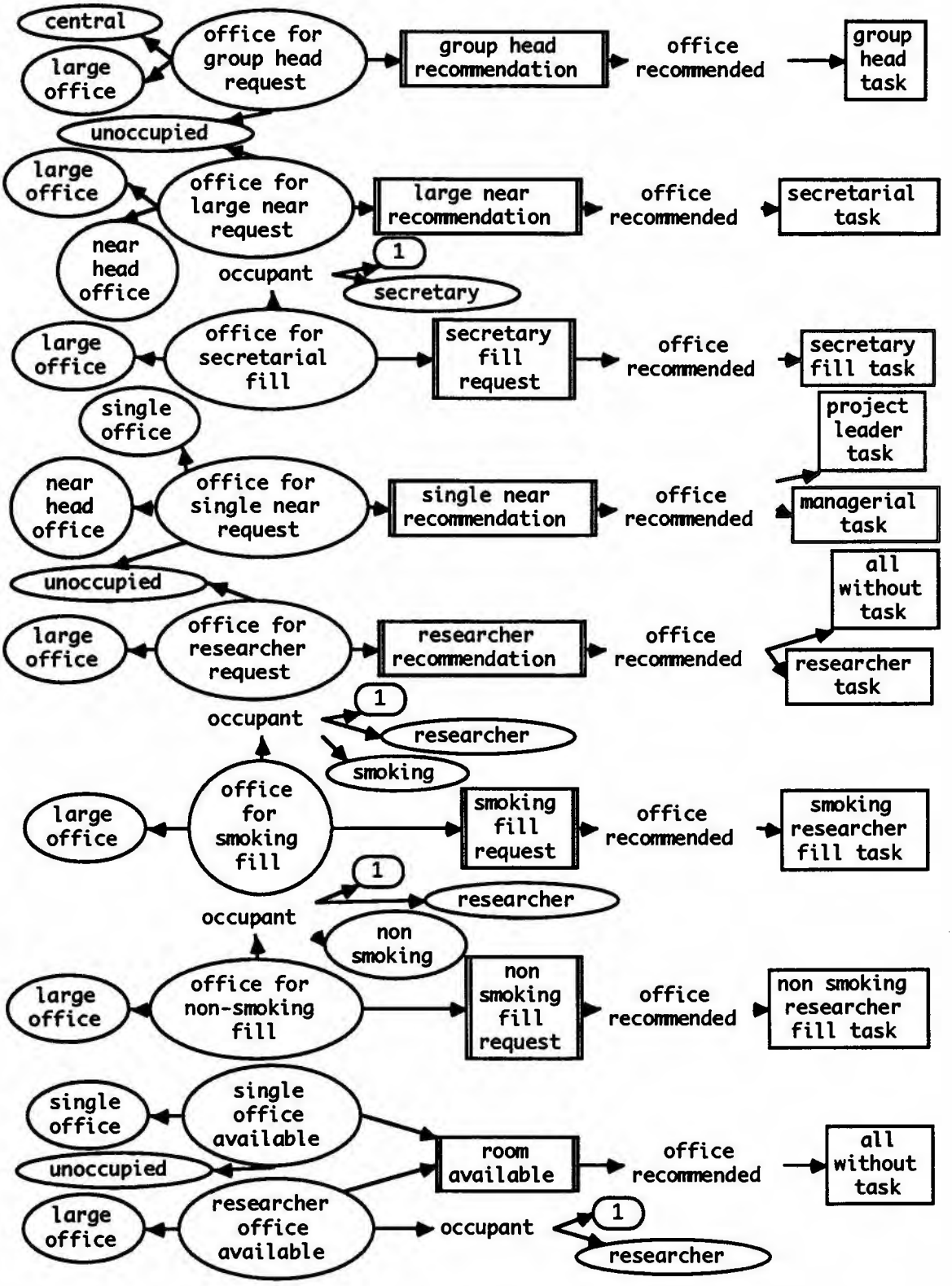
Structure 8: Agenda Mechanism for Room Allocation



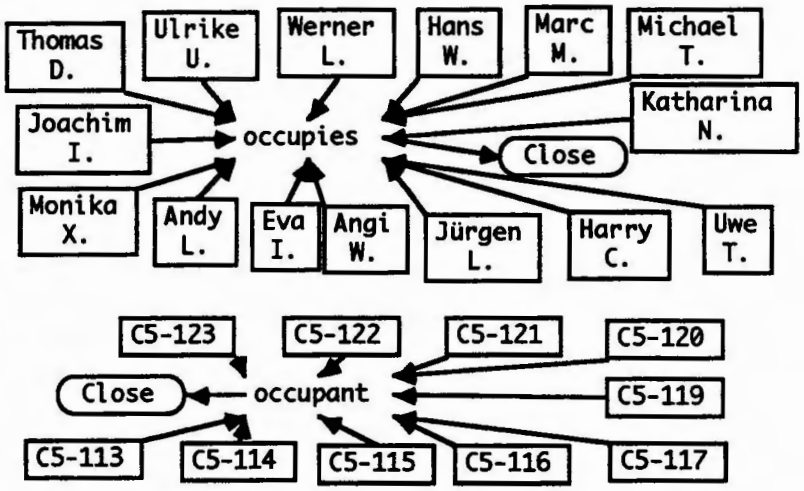
Structure 9: Rule for Room "Near Head Office"



Structure 10: Rules for Employee's Office Requirements



Structure 11: Rules for Office Suitability



Structure 12: Initial State of Employees and Offices

Structure 12 shows an initial state for problem solving when no one has been allocated a room and no rooms are occupied (the roles have been closed with no fillers). Other initial states are possible corresponding to partial allocation. All of the knowledge structures required for the room allocation problem are in this paper. They provide the user interface for domain, problem-solving and particular problem description. It remains to describe the user interface for problem solving. KRS is a server providing problem-solving capabilities, and KWrite and KDraw provide knowledge entry and editing sub-systems for creating a knowledge base. It is possible to use either textual or graphic querying of KRS to solve a particular problem. However, for the Sisyphus example the use of HyperCard to provide an open architecture user interface to KRS was demonstrated. Figure 3 shows the overall architecture.

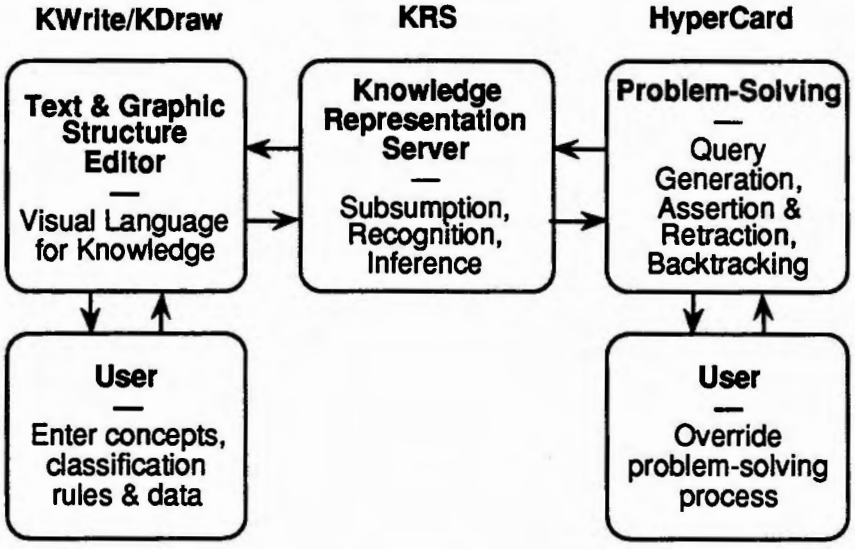


Figure 3 System architecture, functions and user interaction

HyperCard communicates with KRS, as do KDraw and KWrite, through Apple's System 7 inter-application communication protocol. This allows a server instantiation to run anywhere on a network and be accessed locally or remotely. The electronic version of this document serves as the knowledge base for KRS, and the visual knowledge structures shown are compiled directly into concepts, roles, individuals and rules in KRS.

Figure 4 shows the initial screen in HyperCard. The fields at the lower left list the knowledge bases embedded in this paper that will be used in problem solving. The user clicks on the "Solve" button to load these into a KRS server and commence inference. A sequence of recommended room allocations is then shown. If the problem is under-determined, this will involve some choice. Figure 5 shows the screen when the group head has been allocated and now the secretarial allocations are possible and top priority.

The person and room recommended for allocation are highlighted but users can over ride these by clicking on their own choices if they wish. Clicking on the "Allocate" button sends a message to KRS allocating the highlighted room to the highlighted person. The windows on the right in Figure 5 show all the employees without rooms and all the rooms available so that the user can override the recommendation process completely if desired. The user can also go to the screen shown in Figure 6 at any time, see the allocations already made, select any number of them and retract them. The KRS truth maintenance system automatically undoes any conclusions based on retracted data.

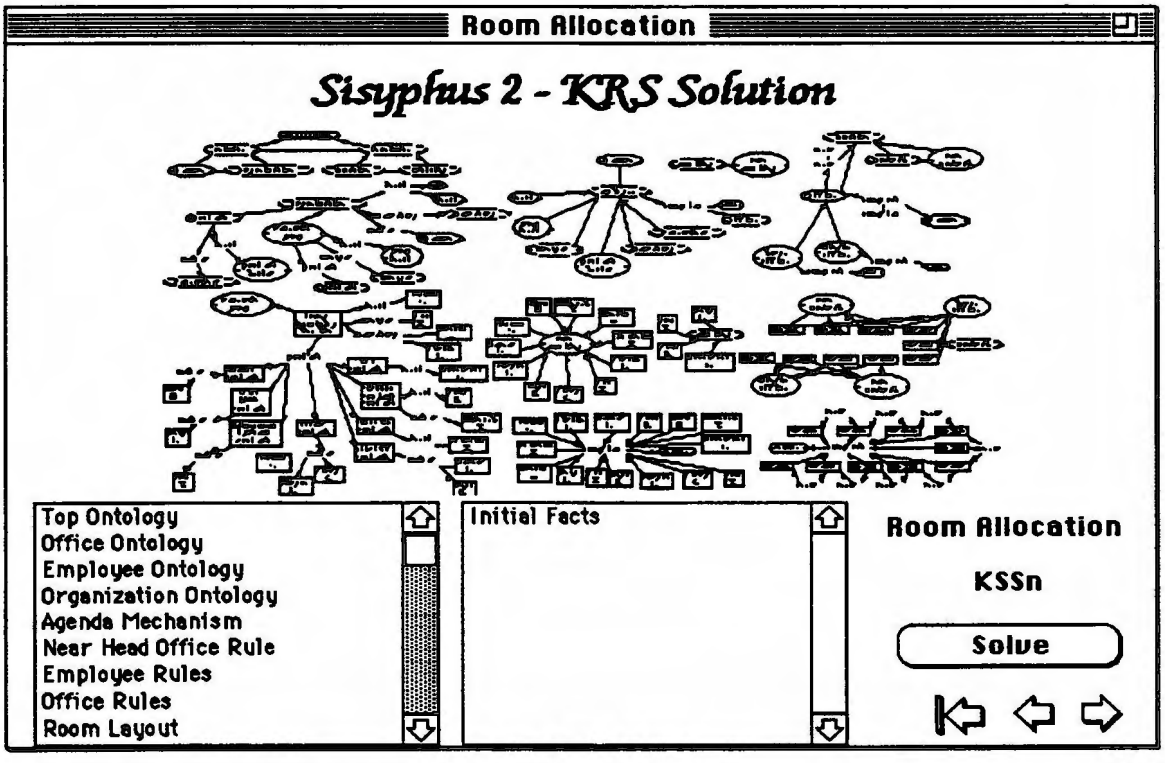


Figure 4 Initial HyperCard screen

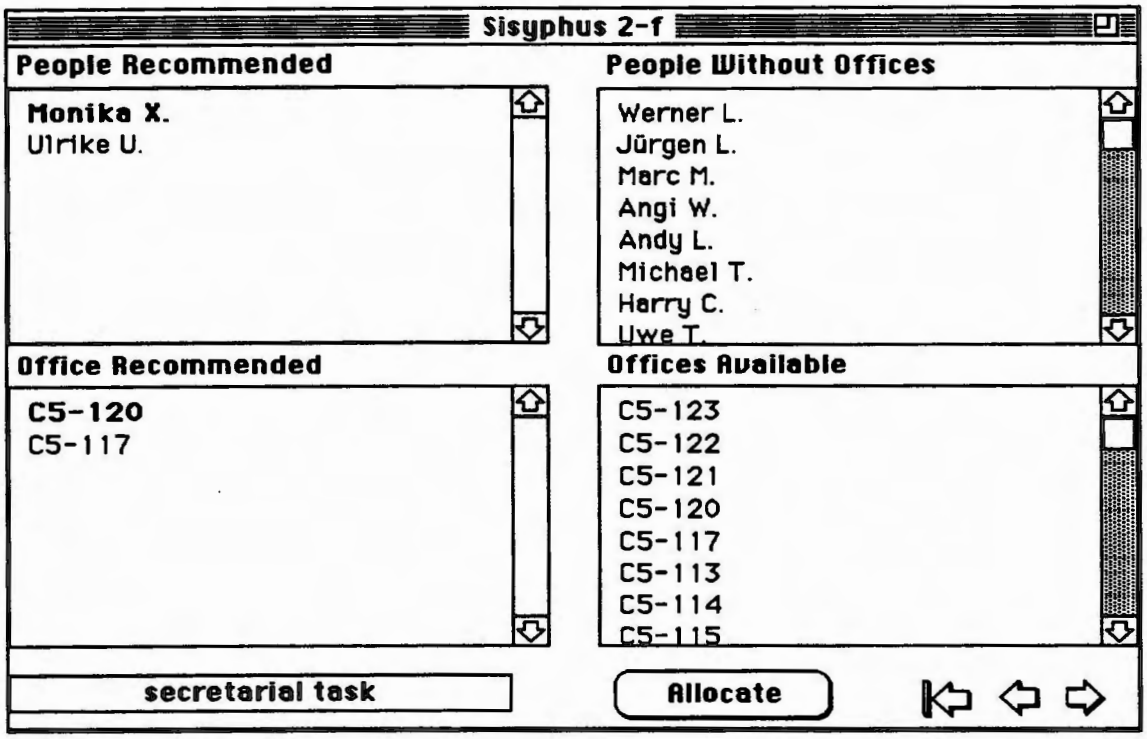


Figure 5 Secretarial room allocation recommendations

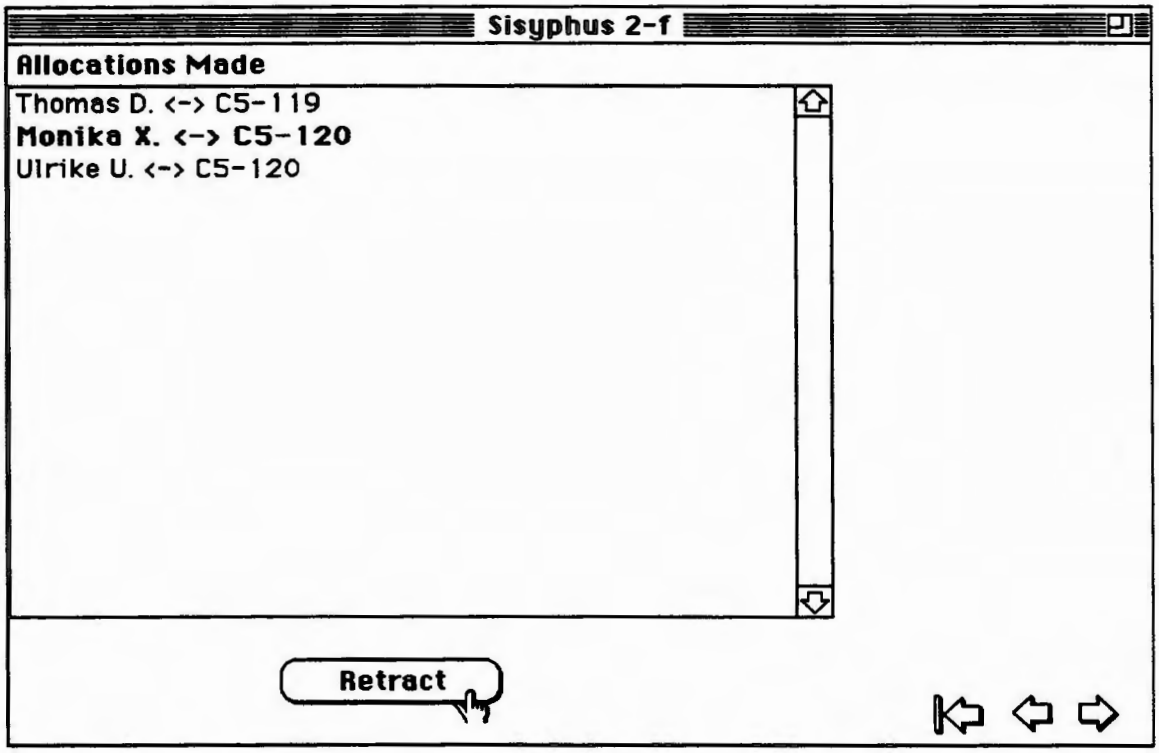


Figure 6 Allocations and retraction

4 CONCLUSIONS

This paper is written in a document production tool that appears to a user as a word processor but also acts as an expert system shell with frame and rule representations supporting deductive inference. The electronic version of the document is active, providing typographic text and page layout facilities, versioning, hypermedia sound and movies, hypertext links, and knowledge structures represented in a visual language. It can be read as a hypermedia document and also interrogated as a knowledge-based system for problem-solving. The paper version of the document, which you have just read, is produced by printing the electronic version. It loses its active functionality but continues to act as a record of the knowledge in the document. The overall technology has been developed as an alternative approach to the dissemination of knowledge bases. It also provides a different interface to knowledge-based systems that emulates document interfaces with which many users are already familiar.

In addition to demonstrating the knowledge document system, this paper has reported some experience in the design and implementation of a lightweight, object-oriented knowledge representation server, and its application to organizational modeling and problem solving. The primary user interface is through a formal visual language implemented simply and naturally as a drawing environment on graphic workstations. The open architecture implementation of the server allows it to be integrated with existing applications, such as corporate database and accounting systems, and also allows additional functionality to be added through self-contained modules requiring no changes in the kernel system.

The problem solving example given has shown how knowledge entered visually can be used to model organizational structures in a way that is simple and natural, and leads directly to operational problem solving. The process described here has the following features:

- The knowledge and data structures are totally overt and easily edited
- The knowledge document format allows a single instance of them to be disseminated both as an active knowledge base and as a passive paper report
- The visual language allows knowledge associated with structures such as the organization chart and room layout to be presented very naturally
- The problem solving strategy is incremental and can be applied to extend an existing partial solution
- The arbitrary choices that arise in undetermined problems can be made by the system or made by a person with, perhaps, additional considerations in mind
- Condensed and understandable information is available through the agenda items to support an attractive presentation of the problem solving process to the user
- The solution developed is a highly generic problem solving strategy
- The various components of the solution may be envisioned as coming from different archives in a corporate knowledge repository

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 that have influenced the research described. We are particularly grateful to Marc Linster, the GMD and the Esprit Reflect Project for making the dataset available. The research reported here would not have been possible without access to the word processing and page makeup software developed by Gary Crandall of Datapak, and much of the significant document processing functionality is a consequence of his work.

REFERENCES

Abrett, G. & Burstein, M.H. (1988) The KREME knowledge editing environment. In Boose, J.H. & Gaines, B.R., Eds. Knowledge Acquisition Tools for Expert Systems. pp.1-24. London, Academic Press.

Borgida, A., Brachman, R.J., McGuiness, D.L. & Resnick, L.A. (1989). CLASSIC: a structural data model for objects. Clifford, J., Lindsay, B. & Maier, D., Eds. Proceedings of 1989 ACM SIGMOD International Conference on the Management of Data. pp.58-67. New York: ACM Press.

Brachman, R.J. (1977) What's in a concept: structural fondations for semantic nets. International Journal of Man-Machine Studies 9, 127-152.

Brachman, R.J. (1979). On the epistemological status of semantic nets. In Findler, N.V., Ed. Associative Networks: Representation and Use of Knowledge by Computers. pp.3-50. New York: Academic Press.

Brachman, R.J. & Schmolze, J. (1985). An overview of the KL-ONE knowledge representation system. Cognitive Science, 9(2) 171-216 (April-June).

Cercone, N. & Schubert, L. (1975). Towards a state-based conceptual representation. Proceedings of AAAI75. pp.83-90. Los Altos: Morgan Kaufmann.

Crandall, G. (1990). Word Solution Engine Programmer's Manual. Vancouver, Washington, DataPak.

Fahlman, S.E. (1979). NETL: A System for Representing and Using Real-World Knowledge. Cambridge, Massachusetts: MIT Press.

Gaines, B.R. (1988a). Structure, development and applications of expert systems in integrated manufacturing. Kusiak, A., Ed. Artificial Intelligence Implications for CIM. pp.117-161. Bedford, UK: IFS Conferences.

Gaines, B.R. (1988b). Knowledge acquisition systems for rapid prototyping of expert systems. INFOR, 26(4), 256-285 (November).

Gaines, B.R. (1990). Knowledge support systems. Knowledge-Based Systems 3(3) 192-203.

Gaines, B.R. (1991a) Empirical investigation of knowledge representation servers: design issues and applications experience with KRS. AAAI Spring Symposium: Implemented Knowledge Representation and Reasoning Systems. pp. 87-101. Stanford (March)—SIGART Bulletin 2(3), 45-56 (June).

Gaines, B.R. (1991b). Integrating rules in term subsumption knowledge representation servers. AAAI'91: Proceedings of the Ninth National Conference on Artificial

Intelligence. pp.458-463. Menlo Park, California: AAAI Press/MIT Press (July).

Gaines, B.R. (1991c). An interactive visual language for term subsumption visual languages. IJCAI'91: Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence. San Mateo, California: Morgan Kaufmann.

Gaines, B.R. (1991). Organizational modeling and problem solving using an object-oriented knowledge representation server and visual language. COCS'91: Proc. of Conference on Organizational Computing Systems. pp.80-94. ACM Press.

Gaines, B.R. & Linster, M. (1990). Integrating a knowledge acquisition tool, an expert system shell and a hypermedia system. International Journal of Expert Systems Research and Applications 3(2) 105-129.

Gaines, B.R., Rappaport, A. & Shaw, M.L.G. (1989). A heterogeneous knowledge support system. Boose, J.H. & Gaines, B.R., Eds. Proceedings of the Fourth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop. pp.13-1-13-20. Banff (October).

Gaines, B.R. & Shaw, M.L.G. (1990) Cognitive and logical foundations of knowledge acquisition. Boose, J.H. & Gaines, B.R. (Eds) Proceedings of the Fifth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop. pp. 9-1-9-25. Banff (November).

Glinert, I.P., Ed. (1990) Visual Programming Environments: Paradigms and Systems. Los Alamitos, California: IEEE Computer Society Press.

Kindermann, C. & Quantz, J. (1989) Graphics-oriented user interfaces for KL-ONE. KIT Internal Report 23. Technical University of Berlin.

Linster, M., Ed. (1991) Sisyphus Working Papers Part 2: Models of Problem Solving. EKAW91, Glasgow: University of Strathclyde.

Nosek, J.T. & Roth, I. (1990) A comparison of formal knowledge representations as communication tools: predicate logic vs semantic network. International Journal of Man-Machine Studies 33, 227-239, 1990.

Quillian, M.R. (1968). Semantic memory. Minsky, M, Ed. Semantic Information Processing. pp.216-270. Cambridge, Massachusetts: MIT Press.

Schmolze, J. (1983). KLONEDRAW—a facility for automatically drawing pictures of KL-ONE networks. Research in Knowledge Representation for Natural Language Understanding. pp.41-44. Report No.5421, Cambridge, Massachusetts: Bolt Beranek and Newman Inc.

Shaw, M.L.G. & Gaines, B.R. (1987). KITTEN: Knowledge Initiation & Transfer Tools for Experts & Novices. International Journal of Man-Machine Studies, 27, 251-280.

Voß, A., Karbach, W., Drouven, U., Lorek, D. & Schuckey, R. (1990) Operationalization of a synthetic problem. ESPRIT Basic Research Project P3178 REFLECT Task I.2.1 Report (July).

Watanabe, H. (1989). Heuristic graph displayer for G-BASE. International Journal of Man-Machine Studies, 30(3) 287-302 (March).

Woods, W.A. (1975) What's in a link: Foundations for semantic networks. Bobrow, D.G. & Collins, A.M. (Eds) Representation and Understanding: Studies in Cognitive Science. pp.35-82. New York: Academic Press.

IMPACT OF A GDSS FOR MULTIPLE EXPERT KNOWLEDGE ACQUISITION

Martha Grabowski
Le Moyne College
Syracuse, New York USA 13214
(315) 445-4427
email: grabowsk@lemoyne.bitnet
and
Department of Decision Sciences and Engineering Systems
Center for Industrial Innovation 8211
Rensselaer Polytechnic Institute
Troy, NY 12180 USA
(518) 276-2954

ABSTRACT

In investigating the use of computer-based group decision support systems (GDSS) during knowledge acquisition (KA), we investigate groups and group processes, traditional roles and impacts of GDSS, the special requirements of using GDSS for KA, and the concepts of "shared mental models". We conclude with a discussion of the impact of expected and unexpected contributions of a GDSS for KA in a real-time operational control environment.

1. INTRODUCTION

There has been a long-standing problem with inferring the causes of complex behavior. Mental events are not directly observable; they must be inferred from overt behavior. Behaviorists often reject mental events as legitimate scientific concepts [Carroll and Olson, 1987]. More recently, however, developments in cognitive science and artificial intelligence, in which mental events are specifically modeled and found to have measurable correlates in behavior, have brought these concepts back into fashion. Eliciting and representing these mental events (i.e., the process of knowledge acquisition (KA)) are the subjects of this paper. We focus specifically on the mental events that occur as experts perform complex tasks in highly turbulent decision-making settings, using a group decision support system (GDSS) to facilitate knowledge acquisition.

From the point of view of cognitive science, experts base their behavior on stored knowledge about particular sequences of actions, on general rules about how to accomplish certain tasks, or on a mental model (an underlying understanding of how a system works). Knowing what an expert knows or expects has implications for systems design. For instance, if this knowledge could be adequately elicited and represented, a system could be designed to fit the user's goals in

accomplishing tasks or could display enough of how it works to make accomplishing a task easy to understand.

At present, there is no standard way of describing what a domain expert knows. There are ways of characterizing differences between experts in a domain, although few involve distinctions about how experts go through the process of developing an awareness and understanding of a domain. We explore this area as researchers concerned with the psychology of performance of complex tasks, as well as practitioners who would like to use the information about how an expert thinks about tasks and systems in the design of improved intelligent' software.

No matter the cognitive or decision-making model, three points are clear:

- decision-making is a multi-phased process, encompassing several different cognitive activities,
- group decision-making juxtaposes a multi-phased process (decision-making) with social, cognitive, and psychological group processes, which have been extensively researched, and
- examining how group processes proceed for groups of experts, with and without the provision of a GDSS to facilitate KA, is a topic of current interest.

This paper provides a brief overview of these topics; a more thorough discussion of group KA is given in earlier work [Grabowski, Massey and Wallace, 1992]. We conclude with a discussion of the expected and unexpected contributions of a GDSS for KA in a real-time operational control environment.

2. GROUPS AND GROUP PROCESSES

To exchange information, members of a group must share some common understandings and perceptions of problem-solving activities. Individual mental models contain common aspects of each member's knowledge and that portion of the knowledge that is not common, but that may be communicated [Minsky, 1986]. The latter can be the result of creative processes, fostered by group interaction. Amabile [1983] proposed a model of creativity, consistent with decision-making and problem-solving models [Mead, 1985], which proposes that an individual builds or recalls information and solutions relevant to tasks at hand. These approaches may be algorithmic or heuristic, depending upon the nature of the task and the individual's level of knowledge about the task domain. Task motivation, domain-relevant skills (knowledge, technical expertise), and creativity-relevant skills affect the products of the creative process. Often, groups, which can serve as important sources of divergent ideas, alternatives, and solutions, can facilitate the generation of creative thoughts, approaches, or solutions.

Although many studies cite improved decisions from groups versus individuals (i.e., Turoff and Hiltz, 1982; Steeb and Johnston, 1981; Gallupe, DeSanctis, and Dickson, 1988), there are a number of barriers to effective group decision-making. For instance, *polarization* can occur when an individual's initial response is exaggerated in the group [Baron and Byrne,1991; Brown, 1988; Paulus, 1989]. This phenomenon suggests that groups may produce more risky decisions than individuals, although it depends largely on the nature of the group's initial attitudes and current issues [Isenberg, 1986].

In some group decision tasks, group members may hold back information, or discuss information already known to the group, as opposed to offering new ideas [Baron and Byrne, 1991]. Remedies have been proposed, including reminding individuals of their areas of expertise and breaking and reconvening meetings later so that additional information can be collected. Wanous and Youtz [1986] offer that increasing the diversity of group members can potentially increase the quality of the decisions made, leading to more views being discussed.

Group effectiveness seems to be correlated with *group size*, the *degree of structure of the group process*, and the *type of task [Grabowski, Massey, and Wallace, 1992]*. As a result, managing a group and group processes is as important as the resulting group choices [Forsythe and Buchanan, 1989]. For instance, formal brainstorming techniques have been helpful in large groups, but proved of little use in small groups. Slater [1958] found that for tasks involving decisions based on the evaluation of exchanged information, groups of five or fewer are most effective.

Sometimes, freely interacting groups can be swayed by strong personalities and may rapidly narrow their focus to a few approaches or unduly concentrate on evaluating ideas. This is especially undesirable in domains characterized by widely divergent expert opinions, as in approaches to battle management problems, or in diagnosing equipment failures.

2.1 Group Processes

Some researchers [Mintzberg, et al., 1976] have suggested that individual and group decision-making go through various stages. Tuckman [1965] developed an early, and still widely used, model that outlined five stages of group development: forming, storming, norming, performing, and adjourning. Group process variables are traditionally categorized as input, process, and output variables. *Input variables* refer to the personal characteristics individuals bring to the group, characteristics of the group itself, and factors related to the environment or the task performance situation in which the group is functioning. *Process variables* refer to the dynamics of group interaction or how the group members communicate with each other in order to coordinate their individual actions. *Outcome variables* are those related to how well the group performed the task, as well as other outcomes indicating how the group members feel about each other, and what they have learned about each other as a result of the interaction.

Variables and their interactions are more thoroughly detailed in [McGrath, 1964; Todd and Benbasat, 1987; Foushee and Helmreich, 1988], and in Figure 1.

Some of the least understood and least studied, but perhaps the most important, factors are those that are related to the actual process of group interaction [Todd and Benbasat, 1987; Foushee and Helmreich, 1988]. Questions that proceed from this examination include: what is the nature of the mechanism that allows a group to work together in task performance? How is responsibility delegated? How often do group members communicate with each other? What is the style of their interchange? Do communication patterns vary in the group, depending on their origin?

Since the communications process itself is the most outward and measurable manifestation of the process of group interaction, we examine group communication as a primary variable in this research. Most studies of groups have ignored process variables [Foushee, 1984; Hackman and Morris, 1975], instead concentrating on direct links between input and output factors (performance). In this paper, we focus specifically on the impact of a GDSS for KA on group process variables in high turbulence decision-making situations.

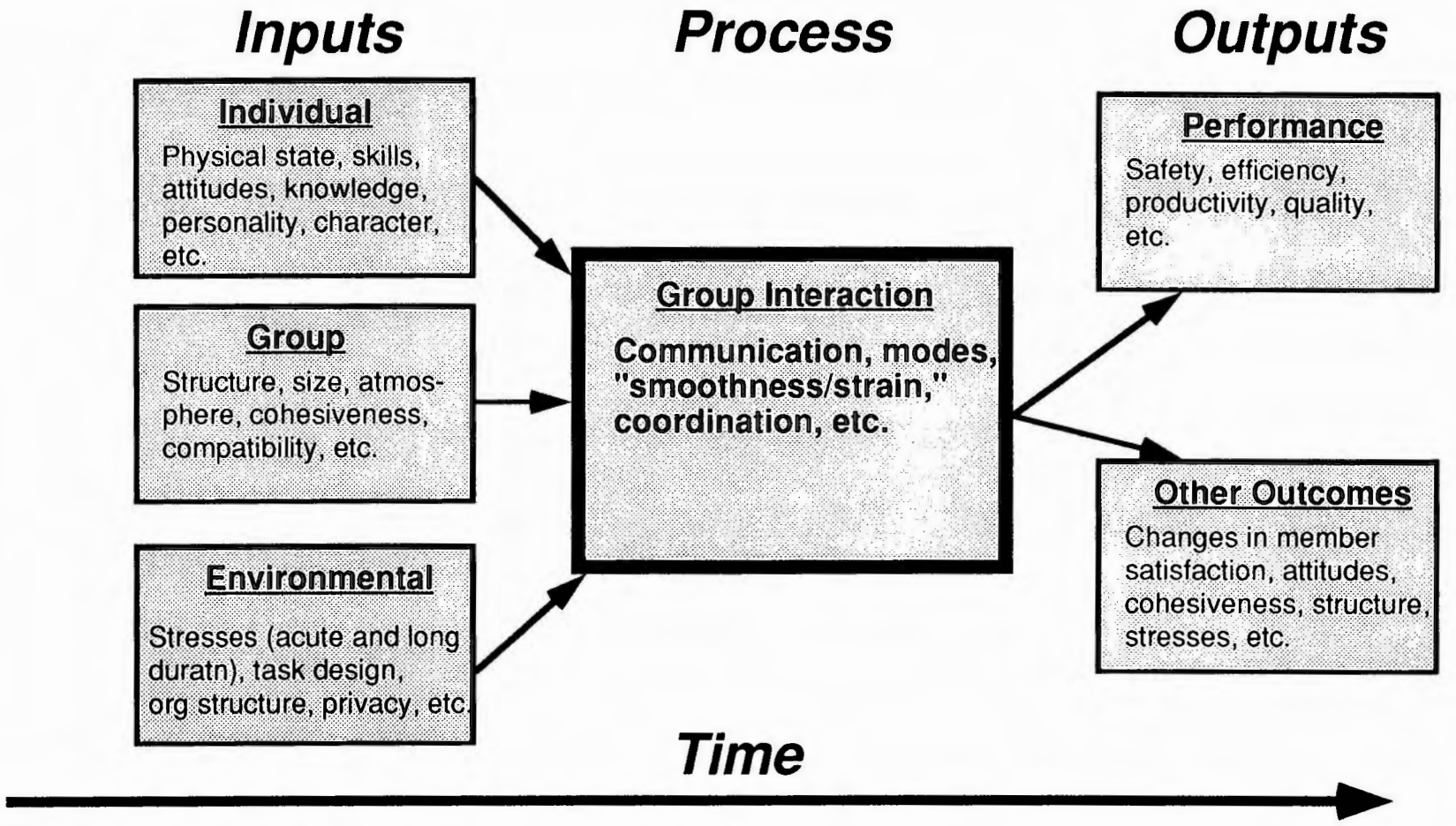
One of the most significant group process variables in high turbulence decision-making situations (i.e., flight crew decision-making) is the *information flow in the group* [Foushee and Helmreich, 1988]. The measurement of relational communication has been utilized over the years by a number of researchers in various paradigms [Bales, 1950]. In studies that have examined the relationship between group process variables and performance effectiveness, careful analyses of the communications process have often proved fruitful. For instance, Lanzetta and Roby [1960] monitored and recorded all communications during a group performance task. Their study found that this particular measure of group interaction predicted task success better than such measures as member knowledge or group skill. They suggest that the way the group "utilizes" its resources and the procedures it employs for communicating essential information are as important, if not more important, than "knowledge" of the problem for determining its performance [p. 146].

Foushee and Manos [1981], in an examination of flight crews during a simulation study, found evidence for higher rates of response uncertainty, frustration or anger, embarrassment, and lower rates of agreement in crews who tended to make more errors. Despite the fact that these correlational data do not allow inferences of causation, some researchers have inferred that the discord related to the commission of errors, whether cause or effect, which could be related to crew coordination deficiencies downstream [Foushee and Helmreich, 1988].

2.2 Eliciting Knowledge from Groups

Several group KA techniques have evolved over time: *nominal group techniques* [Huseman, 1973]; *Delphi techniques* [Linstone and Turuff, 1975]; and *social*

Figure 1 Group Performance Variables



[Adapted fromMcGrath 64; Foushee and Helmreich 88]

judgment analysis [Hammond et al., 1975]. In general, these techniques can be highly structured (i.e., structured workshops) or less structured activities (i.e., focus groups, brainstorming) (Figure 2) [Grabowski, Massey and Wallace, 1992].

Semi-structured group elicitation techniques, which contain software and/or telecommunications capability to provide cues for group elicitation, include *groupware* [Johansen, 1988; Bullen and Bennett, 1990a, 1990b], *interorganizational computer networks* [Rice, 1987; Estrin, 1987], *hypermedia* [Rantanen, 1990], *group writing* [Gallegher and Kraut, 1990], and *teleconferencing* [Fish, Kraut, and Chalfonte, 1990], in decreasing order of group activity structure [Grabowski, Massey, and Wallace, 1992].

The least structured group KA techniques include *brainstorming* and the use of *focus groups*. Focus groups rely on the members' abilities to draw responses from one another through informal discussions among a small group of individuals, ideally five to nine people. These discussions, typically of two hours' duration, are focused by a skilled moderator on a particular topic. An investigation of the use of focus groups as a group KA technique was reported earlier [Grabowski, Massey and Wallace, 1992].

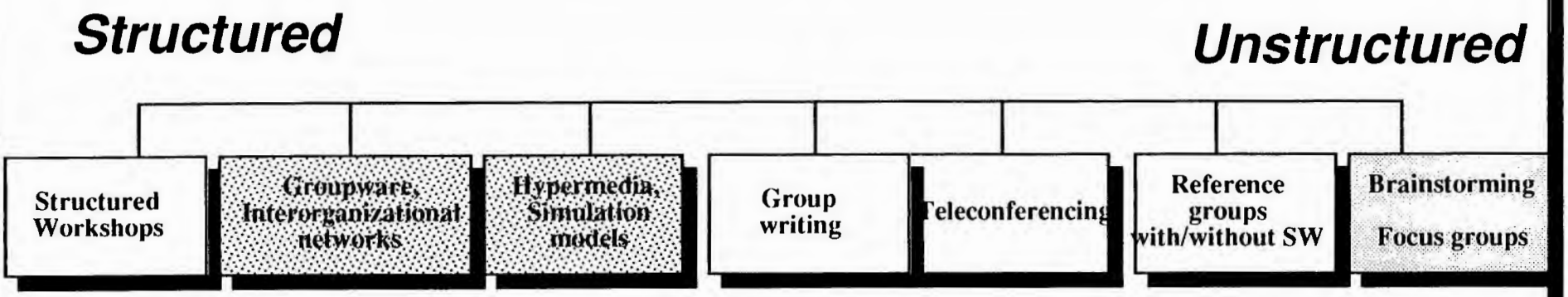
2.3 Group Decision-Making in Shipboard Navigation

In this research, we investigate the use of a GDSS to facilitate KA in a high turbulence decision-making environment. In highly turbulent decision-making, there is continuous "dynamism" [Dess and Beard, 1984] or "volatility" [Bourgeois, 1985], overlaid by sharp and discontinuous change [Meyer, 1982; Sutton, Eisenhardt, and Jucker, 1986]. By extension, high velocity organizations are those in which there is rapid and discontinuous change in market demand, competitors, technology, and/or regulation, such that information is inaccurate, unavailable, or obsolete [Bourgeois and Eisenhardt, 1988]. In this research, we are most concerned with highly turbulent decision-making, rather than organizations, as it is this information which is often elusive in knowledge acquisition. Since information in this environment changes rapidly, it is often difficult to take decisional "snapshots" of the body of knowledge present or used in a decision setting, and thus, knowledge based systems developed for such environments may be obsolete before they are implemented. Exploring the use of a GDSS by decision-makers in such an environment may allow us to capture some of the decisional "snapshots" that have eluded us in previous work [Grabowski and Wallace, in press; ALBM, 1989].

We choose as an example of such an environment the navigation of large tankers in a congested waterway. Three participants are important to the safe navigation of such vessels: the ship's captain, who has overall responsibility for the safe conduct of the vessel; the pilot, who has specialized local knowledge of the port; and the ship's navigating officer, who is the watch officer responsible for the navigation task. Each of these "experts" has important navigation and piloting information which needs to be shared in order for safe and effective navigation

Figure 2 Group Knowledge Acquisition Techniques

Eliciting Knowledge from Groups



to occur. However, the nature of the information and the importance of each participant's knowledge changes during the course of a voyage.

Three types of information are crucial to piloting decision-making: *shiphandling knowledge* (embodied in the captain), *local knowledge* (present in the "knowledge base" of the pilot), and *transit-specific information*, information peculiar to the particular transit underway, on a particular day, in a specified port, with a specific bridge watch team (represented by the pooled knowledge of the captain, pilot, and mate on watch) [Grabowski and Wallace, in press].

Ship's piloting requires reasoning about the three types of information just discussed, in effecting the three primary tasks of ship's piloting: trackkeeping, maneuvering and collision avoidance, and the "practice of good seamanship" [Grabowski and Wallace, in press]. Effecting safe navigation requires a mix of cognitive skills, as trackkeeping requires motor and perceptual skills, while the ethereal "practice of good seamanship" is cognitively complex, requiring a mix of cognitive skills. Ship's piloting is inherently a group process, as the three decision-makers must pool their knowledge in order to produce an effective voyage plan. The decision-making is complicated by an imposed hierarchical organization structure, which places the responsibility for the safe conduct of the vessel squarely on the shoulders of the captain, as advised by the pilot and as executed by the navigating officer.

The three decision makers are inundated with data from a variety of sources, and are required to make crucial piloting decisions in real time [National Research Council, 1990]. These decisions take place in a stressful maritime environment, often after lengthy cargo watches, and are increasingly made by smaller bridge watch teams. These conditions frequently cause information overload, which results from the group's inability to interpret large amounts of data rapidly. The result is the inability to match the data with the appropriate knowledge in time to provide the information needed to perform the piloting task [Grabowski and Wallace, in press]. Thus, understanding and improving group decision processes in this environment can offer significant benefits--to the safe conduct of vessels, to the environment, and to the economic operation of waterborne transportation systems.

3. GROUP DECISION SUPPORT SYSTEMS (GDSS)

Group decision support systems (GDSS) combine computer, communications, and decision technologies to support problem finding, formulation, and solution in group meetings [DeSanctis and Gallupe, 1987; Gallupe, DeSanctis, and Dickson, 1988]. Because decision-making in groups occurs through interpersonal communication, the fundamental goal of a GDSS is to support the exchange of ideas, opinions, and preferences within the group [Gallupe, DeSanctis and Dickson, 1988]. A GDSS that provides communication support features such as anonymous message passing or voting is a *Level 1 GDSS*. When decision modeling or mathematical techniques are added to the system, an enhanced, or

Level 2 GDSS, results. In this paper, we explore the potential contribution of a Level 2 GDSS for KA sessions with multiple experts.

Decision support technology aims to ease the decision-making processes by, among other things, reducing the cognitive load, or mental effort, associated with group meetings. Level 1 GDSS facilitate interpersonal sharing of ideas, opinions, and preferences by providing structure and automation to this process. Even a rudimentary Level 1 GDSS should ease some of the communications burden of open verbal exchange, thus simplifying the decision task [Rao and Jarvenpaa, 1991]. It follows that the more difficult the group decision task is, the greater the potential benefit the GDSS should provide.

Three primary benefits of GDSS are often cited:

- support for improved communication,
- support for increased participation, and
- computational support for tasks [Rao and Jarvenpaa, 1991].

A number of empirical studies of the contribution of GDSS have been conducted [Steeb and Johnston, 1981; Turoff and Hiltz, 1982; Beauclair, 1987]. These laboratory studies have been conducted in "crisis" management settings in which small groups of people have been asked to generate action plans or choose from attractive solutions to the crisis. For some of these studies, software has been made available to decision-making groups for soliciting and aggregating members' ideas or preferences, and the performance of these groups have then been compared to groups without access to the software support. Each of these experiments has demonstrated a performance advantage for groups using the GDSS.

Most GDSS research to date has concentrated on the early phases of decision-making; i.e., the brainstorming and problem formulation stages. Little work has been done in assessing the contribution of GDSS to the alternatives development or "choice" stages of decision-making, or in assessing the "yield" of a GDSS (i.e., the relationship between a GDSS' recommendation(s) and the resulting group decision). This emphasis appears to have more to do with the technological capabilities of current GDSS, rather than reflecting interesting research choices [Rao and Jarvenpaa, 1991]. In this research, we explore the use of a GDSS to capture the alternatives generation and choice phases of highly turbulent decision-making.

4. INTRODUCTION OF A GDSS FOR KNOWLEDGE ACQUISITION

Hackman and Morris [1975] suggest that the best way to effect meaningful change in group performance is to concentrate on input factors. Thus, it is often proposed that group performance strategies can be made more task-appropriate by modifying the group's norms, that member effort and coordination can be

increased by task redesign, and that the level and utilization of group member knowledge and skill can be improved by altering the composition of the group.

In this work, we investigate the use of a GDSS that provides on-line piloting recommendations to ship's masters and pilots piloting large tankers in Valdez, Alaska [Grabowski and Sanborn, in press]. The system hypothesizes multiple alternate courses of action in vessel trackkeeping, maneuvering and collision avoidance problem situations, as an embedded intelligent node in Exxon Shipping Company's ExxBridge integrated bridge system (IBS). The ExxBridge IBS provides a common display for the pilot, captain, and navigating officer; it is the task of this group of experts to pool their knowledge and determine safe voyage plans for the vessel in the waterway. It is the pooled knowledge of this group of experts that we sought to capture with the use of the GDSS.

In this research, we investigate altering *group process factors* in order to understand (and perhaps improve) group KA performance. We hypothesize that the provision of a GDSS will not only support increased communication among group members, provide support for increased participation, and provide computational support for tasks (i.e., traditional contributions of GDSS to group decision-making [Rao and Jarvenpaa, 1991]), but also facilitate the development of a group mental model of the domain, problem, and courses of action, as an aid to enhancing group productivity (i.e., more ideas, better ideas, more well-formulated opinions, etc.) and as a catalyst to surface "more and better expertise." The focus of this research is upon the group performance process as it affects crew effectiveness in multipilot operations. This task is a classic small group performance situation in a high turbulence decision-making environment [Bourgeois and Eisenhardt, 1988].

4.1 Previous Research

In earlier work, we sought to determine what types of heuristics could be elicited from ship's pilots with three "manual" knowledge acquisition techniques: scenarios, simulated familiar tasks, and actual familiar tasks [Grabowski and Wallace, 1992]. The results showed that 30% of the experts' heuristics were common, indicating that these heuristics might form the core of a shared mental model common to all Sandy Hook Pilots transiting New York harbor. This could indicate that a set of piloting heuristics was accessible, regardless of the knowledge acquisition method employed. Conversely, 70% of the expert heuristics were only elicited by a single KA method, giving rise to the suggestion to use a "mix" of KA methods in order to access different "packages" of expert knowledge [Anderson, 1983].

Few descriptions of knowledge acquisition processes have discussed multiple expert interactions. Two notable exceptions are Shaw [1988] and Gaines [1988]. In addition, Grabowski, Massey and Wallace [in press] investigated the hypothesis that face-to-face group interaction with experts in focus groups provided different "thoughts and intuitions" about a task or situation than

individual interviews. Multiple experts were used in a situation assessment context. Focus groups were found to be useful in knowledge acquisition because the interaction between respondents drew more cognition and "rules" to the surface. The subsequent verbatim transcripts of the respondents were analyzed to discover the underlying rules and cognitive schema [Massey, 1991].

4.2 Description of the GDSS

To assist navigational officers in effecting safe passages through congested waters, particularly in conditions of fatigue or stress, *integrated bridge systems* (IBS) are being developed by a number of nations: the United States, Norway, West Germany, Japan, and the United Kingdom [Grove, 1989]. These integrated bridges project the ship's wheelhouse as the operational center for navigational and supervisory tasks aboard ship; the bridges in many cases become `ship's operations centers', incorporating controls and monitors for all essential vessel functions--navigation, engine control, and communications.

The underlying rationale for IBS is the thought that integrating data traditionally obtained manually from several navigational instruments into a single workstation will free the watch officer's time and attention so that more attention can be focused on safe and efficient navigation. Thus, in IBS, many routine navigational tasks, such as chart updating, position plotting, and steering are automated, and the IBS is designed as a federation of subsystems supporting vessel navigation, communications, steering, administration, collision avoidance, safety, and ships' systems monitoring and control functions [Grove, 1989].

Introduction of bridge automation can be effective in combatting shipboard stress and fatigue, by off-loading non-critical monitoring tasks from the human decision-maker. Thoughtful introduction of bridge decision aiding technology has been shown to alleviate stress and fatigue on watch by embedding simulation and functional team training capabilities in automated systems, and providing non-distracting, stimulating exercises [National Research Council, 1990].

The GDSS used in this research is the ExxBridge IBS, with an embedded piloting expert system, which supports the reasoning tasks of the bridge watch team when the vessel is in close waters. The piloting expert system represents internally local, transit-specific, and shiphandling knowledge; supports the appropriate level of piloting cognitive skills; and provides recommendations for effecting the piloting tasks of trackkeeping, maneuvering and collision avoidance, and the practice of good seamanship [Grabowski and Wallace, in press].

The GDSS also provides mental rehearsal capabilities, allowing masters, mates, and pilots to practice typical and atypical inbound and outbound transits. Off-line, the decision aid's voyage planning capability mirrors the way senior ship's officers and pilots encourage their junior officers to plan and mentally rehearse transits prior to assuming a watch. Thus, the GDSS with an embedded piloting decision aid could be expected to impact piloting in two ways: by producing

improved vessel performance (trackkeeping) and *improved bridge watch team performance* (maneuvering, collision avoidance, and the practice of good seamanship skills). The issue is how to obtain the knowledge needed to support the cognitive processing of the human controller, the pilot.

4.3 Research Design

In this experiment, we will use the GDSS (the Exxon ExxBridge IBS) with the piloting expert system, as a KA tool. To evaluate the bridge watch team's performance, we will make multiple shipboard runs along the same route, aboard the same vessel, with the same ship's crew, and the same maritime pilot aboard. Some of these runs will include the use of the GDSS with the piloting expert system, and some will not. In addition to traditional statistical measures of vessel trackkeeping performance (deviation from the course centerline, offset error, swept path, etc.), we will also examine the process variables in the task, concentrating on the expert information elicited during the transit with the GDSS and without the GDSS. These process variables include communication patterns, number and types of communications, communications structure, etc., after Foushee and Helmreich [1988]. We will process code the information elicited during the transit, following Hirokawa and Pace [1983], and we will characterize the information elicited in both types of transits, following the same coding scheme.

In both types of transits, we will examine group process variables in order to characterize the information elicited. As a control, we will use as a baseline two years' worth of KA data elicited on the same vessel, with the same crew, the same pilot, over the same transit. In this way, we will be able to benchmark the old KA performance standard (KA using interviews, protocol analysis, and observation), and compare it to similar (if not identical) KA processes, with the addition of a GDSS for KA.

5. EXPECTED CONTRIBUTIONS

We anticipate a number of contributions from this research, some expected, some not as expected. These contributions include an improved understanding of group processes, improved group decision-making in highly turbulent settings, an improved ability to articulate shared mental models, and perhaps movement toward the ability to generate and develop 'quality' knowledgebases.

5.1 Improved Understanding of Group Processes

Solutions to problems are discovered through devising, specifying, and following combinations of procedures that might achieve specific objectives. Problem-solving involves such tasks as specifying how to perform the expert(s)' tasks. In general, problem solving produces the invention or design of multiple alternative explanations for the functioning (or disfunctioning) of a partially understood

system, or the formulation of answers by diverging as little as possible from seemingly appropriate and well-known rules.

Often referred to as "convergent thinking," such group activity is thought to be at its best when organized and highly systematized. However, the paucity of rules specifying what constitutes key information or what is the essential information to be structured typically makes this type of task most puzzling to organize for a group. Deep knowledge of the system being studied and the nature of the expert system building task at hand is necessary to structure appropriate group activities. Thus, the most critical phases of conceptualization and formalization are often difficult to support with group techniques, unless the group is led by a skilled facilitator with significant understanding of the system building process [Richardson, et al., 1990]. In this research, we assess the contribution of a GDSS in eliciting alternative courses of action in a highly turbulent setting. We hypothesize that the GDSS may provide a structured way of exploring unstructured courses of action, and may also serve as a communications vehicle, facilitating the development of more effective decision-making teams.

We anticipate that this research will contribute to our understanding of group processes in effective decision-making teams. We know that "high error" crews typically have breakdowns in the areas of communication, team coordination, and team integration [Foushee and Helmreich, 1988]; by better understanding these processes, we hope to enhance the effectiveness of these teams.

5.2 Improved Group Decision-Making in High Turbulence Decision Settings

Decision settings in the future will be characterized by more and increasing knowledge, more and increasing complexity, and more and increasing turbulence [Huber, 1984]. As a result, we can expect that information technologies to increase the efficiency and effectiveness of decision-oriented meetings will be of particular significance. As Huber [1984] points out, a significant increase in decision group efficiency and effectiveness can be achieved by creatively integrating command and control (C2) technologies into decision group technologies. These applications have already been developed for teleconferencing, video conferencing, and electronic mail services for aiding distributed decision groups [Johansen, Vallee and Spangler, 1979; Martin, 1981; Hiltz and Turoff, 1978]; their integration into reasoning and decision systems is a natural evolution.

These advances become even more significant as *crisis decision groups* [Smart and Vertinsky, 1977; Bronner, 1982] are introduced into organizations. Thus, if these crisis decision groups or 'tiger teams' are to be deployed in organizations of the future, understanding how these groups function effectively and how to manage them is important. Improving the efficiency and effectiveness of these decision-making groups in organizations is a priority issue in highly turbulent

environments; this research will focus attention on eliciting the "best practices" knowledge of decision-making groups, and representing that information within a KBS as a model for organizations of the future.

These findings are particularly important for *organizations* who are thrust into highly turbulent decision settings, often against their will. Exxon Shipping Company in the post *Valdez*-era is a case in point. Normally, shipping companies, and particularly tanker operations, are conservative decision-making organizations. However, catastrophic organizational, environmental, and litigious events (such as that of the *Valdez* grounding) can catapult a normally staid organization with (semi) rational decision-making into a highly turbulent organization. As developers of knowledge-based systems to be deployed to address some of the causes of the *Valdez* grounding, we were able to run a three-year case study, eliciting piloting knowledge from ship's captains and pilots at a micro level, while gathering meta-level decision-making data (also useful in our knowledge acquisition sessions) at the same time. With this research, we are able to investigate the usefulness of a GDSS for knowledge acquisition at a micro and a macro level, in a highly turbulent decision-making environment. We posit that our observations are extensible to highly turbulent decision-making organizations at large.

5.3 Improved Articulation of Shared Mental Models

Mental models [Rouse and Morris, 1986] represent a user's knowledge of how a system works, what its components are, how they are related, what the internal processes are, and how they affect the components. It is this conceptualization that allows a user not only to construct actions for novel tasks, but also to explain why a particular action produces the result that it does [Carroll and Olson, 1987]. Mental models are most commonly used to refer to a cognitive representation of a physical system or software, with some plausible cascade of causal associations connecting the input to the output [Carroll and Olson, 1987]. Accordingly, a mental model of a system is a rich and elaborate structure, reflecting the decision-maker's understanding of what the system contains, how it works, and why it works that way. It can be conceived of knowledge about the system sufficient to permit the user to mentally try out actions before choosing one to execute. A key feature of a mental model is that it can be "run" with trail, exploratory inputs and observed for its resultant behavior [Sheridan et al., 1986].

Human performance analyses have been well developed in vehicular control (i.e., aircraft, ship, automobile) and target pursuit tasks. Many of these analyses explicitly hypothesize a mental model of the system being operated (i.e., Baron and Levison, 1980; Jagacinski and Miller, 1978; Pew and Baron, 1983; Veldhuyzen and Stassen, 1977). In these cases, the mental model is used to anticipate the response of a dynamic system and hence to overcome the deleterious effects of time delays either from other humans or from hardware. These models have produced good descriptions and predictions of human performance.

The problem, from the knowledge acquisition viewpoint, is that articulating individual mental models is difficult enough; eliciting group mental models is even more so. In this research, we assess the contribution of a GDSS to the articulation of shared mental models among experts in a domain, in the hope that a clearer understanding of the development, articulation, and modification of these models results.

6. IMPLICATIONS FOR PRACTICAL KNOWLEDGE ACQUISITION

In this experiment, we explore the use of a GDSS for multiple expert knowledge acquisition in a high turbulence decision setting. In general, *semi- and ill-structured problem domains* require the alternative perspectives of multiple experts for both knowledge elicitation and validation, since when a problem is not well-structured, one needs to communicate with others to determine what information is relevant [Turoff and Hiltz, 1982]. Problem definitions and problem- solving approaches for semi- and ill-structured tasks generally require some degree of creativity, as problem solutions and paths to solutions are often not clear. Knowledge, imagination, logic, intuition, and constructive evaluation are needed to discover new connections between ideas. Thus, the use of a GDSS in constructing expert systems in semi- and ill-structured problem domains must facilitate creative processes in order to elicit useful knowledge in that domain. We will explore the elicitation of this 'deep' knowledge, and its relation to shared mental models, in future work.

Practically speaking, it is often difficult enough to construct the expert system under development, without contemplating the development of an additional, or parallel, GDSS to be used as a KA tool. However, in many organizations, GDSS exist which codify multiple sources of existing knowledge; many times, these GDSS were laboratory or throw-away prototypes that may have accompanied the development of system specifications, requirements documents, or requests for proposals for the desired system. In any case, these systems exist in many organizations. This research exhorts system builders to contemplate use of these GDSS in a KA context, not only to tap into the knowledge contained therein, but also to assist in assessing (1) the role of group processes in the organization, (2) the nature of group decision-making in highly turbulent decision settings, and (3) the role of shared mental models in group decision-making within that organization.

REFERENCES

DARPA/US Army AirLand Battle Management (ALBM) program. *FIRES-D Functional Requirements Report*. June 1989.

Amabile, T.M., *The Social Psychology of Creativity*, New York: Springer-Verlag, 1983.

Anderson, J.R., *The Architecture of Cognition*, Cambridge, MA: Harvard University Press, 1983.

Bales, R.F., *Interaction process analysis: Theory, research, and applications*, Reading, Massachusetts: Addison-Wesley Publishing Company, 1950.

Baron, R.A., and Byrne, D., *Social Psychology: Understanding Human Interaction*, 6th edition, Boston, Massachusetts: Allyn and Bacon, 1991, pp. 457-467.

Baron, S., and Levison, W. H., "The Optimal Control Model: Status and Future Direction," *Proceedings of the IEEE Conference on Systems, Man, and Cybernetics*, Cambridge, Massachusetts: IEEE Press, 1980.

Beauclair, R., "An Experimental Study of the Effects of Group Decision Support System Process Support Applications in Small Group Decision Making," Ph.D. Thesis, Indiana University, 1987.

Bourgeois, L. J., "Strategic Goals, Environmental Uncertainty, and Economic Performance in Volatile Environments," *Academy of Management Journal*, 28:3, September 1985, pp. 548-573.

Bourgeois, L. J. and Eisenhardt, K. M., "Strategic Decision Processes in High Velocity Environments: Four Cases in the Microcomputer Industry," *Management Science*, 34:7, July 1988, pp. 816-835.

Bronner, R., *Decision-Making Under Time Pressure*, Lexington: DC Heath, Inc., 1982.

Brown, R., *Group Processes*, Worcester, UK: Billing and Sons, Ltd., 1988, pp. 124-164.

Bullen, C.V. and Bennett, J.L., "Groupware in Practice: An Interpretation of Work Experiences," CISR Working Paper No. 205, MIT Center for Information Systems Research, Cambridge, Mass, 02139, 1990a.

Bullen, C.V. and Bennett, J.L., "Learning from User Experience with Groupware," *Proceedings of the Conference on Computer-Supported Cooperative Work,* Los Angeles, 7-10 October 1990b, pp. 291- 302.

Carroll, J.M. and Olson, J.R., (eds.), *Mental Models in Human-Computer Interaction*, National Research Council, Washington, DC: National Academy Press, 1987.

DeSanctis, G. and Gallupe, B., "A Foundation for the Study of Group Decision Support Systems," *Management Science*, 33:5, May 1987.

Dess, G.D. and Beard, D.W., "Dimension of Organizational Task Environments," *Administrative Science Quarterly*, 29, 1984, pp. 52-73.

Estrin, D. (1987), "Interconnections of private networks," *Telecommunications Policy*, September 1987, pp. 247-258.

Fish, R.S., Kraut, R.E., and Chalfonte, B.L., "The VideoWindow System in Informal Communications," *Proceedings of the Conference on Computer-Supported Cooperative Work*, Los Angeles, 7-10 October 1990.

Forsythe, D., and Buchanan, B., "Knowledge Acquisition for Expert Systems: Some Pitfalls and Suggestions," *IEEE Transactions on Systems, Man, and Cybernetics*, Special Issue on Perspectives in Knowledge Engineering, 19:3, May/June 1989, pp. 435-442.

Foushee, H.C., "Dyads and triads at 35,000 feet: Factors affecting group process and aircrew performance," *American Psychologist*, 39, 1984, pp. 885-893.

Foushee, H.C.,and Helmreich, R.L., "Group Interaction and Flight Crew Performance," *Human Factors in Aviation*, New York: Academic Press, 1988.

Foushee, H.C., and Manos, K.L., "Information transfer within the cockpit: problems in intracockpit communications," in C.E. Billings and E.S. Cheaney (eds.), *Information Transfer Problems in the Aviation System*, NASA Technical paper 1875; Moffett Field, CA: NASA Ames Research Center, 1981, pp. 63-71.

Gaines, B.R., "Second Generation Knowledge Acquisition Systems," *Proceedings of the European Knowledge Acquisition Workshop*, John Boose, Brian Gaines, and Marc Linster (eds.), Bonn, Germany, 19-23 June 1988, pp. 17-1 to 17-14.

Gallegher, J. and Kraut, R.E., "Computer-Mediated Communication for Intellectual Teamwork: A Field Experiment in Group Writing," *Proceedings of the Conference on Computer-Supported Cooperative Work*, Los Angeles, 7-10 October 1990, pp. 65-78.

Gallupe, R.B., DeSanctis, G., and Dickson, G.W., "The Impact of Computer-based Support on the Processes and Outcomes of Group Decision-Making," *MIS Quarterly*, 12:2, June 1988, pp. 277-296.

Grabowski, M. R., and Sanborn, S. D., "Knowledge Representation and Reasoning in a Real-Time Operational Control System: The Shipboard Piloting Expert System," *Decision Sciences*, in press.

Grabowski, M.R. and Wallace, W.A., "Knowledge Acquisition for Real Time Operational Control Systems: An Empirical Assessment of Three Methods," Department of Decision Sciences and Engineering Systems, Rensselaer Polytechnic Institute, Technical Report 37-89-199, January 1992.

Grabowski, M.R. and Wallace, W.A., "An Expert System For Maritime Pilots: Its Design and Assessment Using Gaming," *Management Science*, in press.

Grabowski, M.R., Massey, A.P., and Wallace, W.A., "Focus Groups as a Group Knowledge Acquisition Technique," *Knowledge Acquisition,* 4, (1992).

Grove, T.W., "US Flag Ship of the Future: concepts, features and issues," in *Proceedings of the Society of Naval Architects and Marine Engineers' Ship Technology and Research Symposium,* New Orleans, April 1989.

Hackman, J.R., and Morris, C.G., "Group tasks, group interaction process, and group performance effectiveness: A review and proposed integration," in L. Berkowitz (ed.), *Advances in experimental social psychology*, New York: Academic Press, 8, 1975, pp. 45-79.

Hammond, K.R., Stewart, T.R., Brehmer, B., and Steinmann, D.O., "Social Judgment Theory," In M.F. Kaplan and S. Schwartz (eds.), *Human Judgment and Decision Processes in Applied Settings*, New York: Academic Press, 1975.

Hiltz, S. and Turoff, M., *The Network Nation: Human Interaction via Computer*, Reading, Massachusetts: Addison-Wesley Publishing Company, 1978.

Hirokawa, R., and Pace, R., "A Descriptive Investigation of the Possible Communication-Based Reasons for Effective and Ineffective Group Decision-Making," *Communications Monographs*, 50, 1983, pp. 363-80.

Huber, G.P., "The Nature and Design of Post-Industrial Organizations," *Management Science*, 30:8, August 1984, pp. 928-951.

Huseman, R., "The Role of Nominal Groups in Small Group Communication," in R.C. Huseman, D. M. Logue, and D.L. Freshley (eds.), *Readings in Interpersonal and Organizational Communication* 2nd edition, Boston: Hollbrook Press 1973.

Isenberg, D.J., "Group Polarization: A Critical Review and Meta-Analysis," *Journal of Personality and Social Psychology*, 50, 1986, pp. 1141-1151.

Jagacinski, R. J., and Miller, R.A., "Describing the Human Operator's Internal Model of a Dynamic System," *Human Factors*, 20, 1978, pp. 425-433.

Johansen, R., *Groupware: Computer Support for Business Teams*, New York: The Free Press, 1988.

Johansen, R., Vallee, J., and Spangler, K., *Electronic Meetings*, Reading, Massachusetts: Addison-Wesley Publishing Company, 1979.

Lanzetta, J.T. and Roby, T.B., "The relationship between certain group process variables and group problem-solving efficiency," *Journal of Social Psychology*, 52, 1960, pp. 135-148.

Linstone, H. and Turuff, M., *The Delphi Method: Techniques and Applications*, Reading, Massachusetts: Addison-Wesley Publishing Company, 1975.

Martin, J., *Telematic Society: A Challenge for Tomorrow*, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1981

Massey, A.P., "Group Problem Definition: The Impact of Network Structuring Techniques on Group Effectiveness and Interaction," Ph.D. Thesis, Rensselaer Polytechnic Institute, 1991.

McGrath, J.E., *Social psychology: A brief introduction*, New York: Holt, Rinehart and Winston, 1964.

Mead, M., "Creativity by Design: Creativity Enhancing Principles for Decision Support Systems," Ph.D. dissertation, University of Texas at Austin, 1985.

Meyer, A.D., "Adapting to Environmental Jolts," *Administrative Science Quarterly*, 27, December 1982, pp. 515-537.

Minsky, M., *The Society of Mind,* New York: Simon and Schuster, 1986.

Mintzberg, H., Raisinghani, D., and Theoret, A., "The Structure of "Unstructured" Decision Processes," *Administrative Science Quarterly,* 21, 1976, pp. 246-276.

National Research Council (NRC), *Crew Size and Maritime Safety*. Washington: National Academy Press, 1990.

Paulus, P.B., *Psychology of Group Influence*, 2nd edition, Hillsdale, New York: Lawrence Erlbaum Associates, 1989, pp. 330-345.

Pew, R.W., and Baron, S., "Perspectives on Human Performance Modeling," *Automatica*, 19, 1983, pp. 663-676.

Rantanen, J.A., "Hypermedia in knowledge acquisition and specification of user interface for KBS: an approach and case study," *Knowledge Acquisition*, 2, 1990, pp. 259-278.

Rao, V.S. and Jarvenpaa, S.L., "Computer Support of Groups: Theory-based Models for GDSS Research," *Management Sciences*, 37:10, October 1991, pp. 1347-1362.

Rice, R.E., "Computer-mediated communication and organizational innovation," *Journal of Communication*, 37:4, 1987, pp. 65-94.

Richardson, G.P, Vennix, J.A., Andersen, J.D.F., Rohrbaugh, J., and Wallace, W.A., "Eliciting Group Knowledge for Model Building," Rensselaer Polytechnic Institute, Department of Decision Sciences and Engineering Systems, Technical Report No. 37-90-227, March 1990.

Rouse, W.B., and Morris, N.M., "On Looking Into the Black Box: Prospects and Limits in the Search for Mental Models," *Psychological Bulletin*, 100:3, 1986, pp. 349-363.

Shaw, M.L.G., "Problems of Validation in a Knowledge Acquisition System," *Proceedings of the European Knowledge Acquisition Workshop*," John Boose, Brian Gaines, and Marc Linster (eds.), Bonn, Germany, 19-23 June 1988, pp. 5-1 to 5-15.

Sheridan, T.B., Charny, L., Mendel, M.B., and Roseborough, J.B., "Supervisory Control, Mental Models, and Decision Aids," MIT Department of Mechanical Engineering Technical Report, July 1986.

Slater, P., "Contrasting Correlates of Group Size," *Sociometry*, 25, 1958, pp. 129-139.

Smart, C. and Vertinsky, I., "Design for Crisis Decision Units," *Administrative Science Quarterly*, 22, December 1977, pp. 640-657.

Steeb, R., and Johnston, S.C., "A Computer-based Interactive System for Group Decision-Making," *IEEE Transactions on Systems, Man, and Cybernetics*," SMC-11, 8, August 1981, pp. 544-552.

Sutton, R., Eisenhardt, K.M., and Jucker, J., "Managing Organizational Decline: Lessons from Atari," *Organizational Dynamics*, Spring 1986.

Todd, P. and Benbasat, I., "Process Tracing Methods in Decision Support Systems Research: Exploring the Black Box," *MIS Quarterly*, 11:4, 1987, pp. 493-512.

Turoff, M., and Hiltz, S.R., "Computer Support for Group versus Individual Decisions," *IEEE Transactions on Communications,*" COM-20, 1, 1982, pp. 82-90.

Veldhuyzen, W., and Stassen, H.G. The Internal Model Concept: An Application to Modeling Human Control of Large Ships. *Human Factors*, 1977, 19, 478-487.

Wanous, J.P., and Youtz, M.A., "Solution Diversity and the Quality of Group Decisions," *Academy of Management Journal*, 29, 1986, pp. 149-159.

A Translation Approach to Portable Ontology Specifications

Thomas R. Gruber

Knowledge System Laboratory
Stanford University
701 Welch Road, Building C
Palo Alto, CA 94304
gruber@sumex-aim.stanford.edu

Abstract

To support the sharing and reuse of formally represented knowledge among AI systems, it is useful to define the common vocabulary in which shared knowledge is represented. A specification of a representational vocabulary for a shared domain of discourse — definitions of classes, relations, functions, and other objects — is called an ontology. This paper describes a mechanism for defining ontologies that are portable over representation systems. Definitions written in a standard format for predicate calculus are translated by a system called Ontolingua into specialized representations, including frame-based systems as well as relational languages. This allows researchers to share and reuse ontologies, while retaining the computational benefits of specialized implementations.

We discuss how the translation approach to portability addresses several technical problems. One problem is how to accommodate the stylistic and organizational differences among representations while preserving declarative content. Another is how to translate from a very expressive language into restricted languages, remaining system-independent while preserving the computational efficiency of implemented systems. We describe how these problems are addressed by basing Ontolingua itself on an ontology of domain-independent, representational idioms.

1. Introduction

A body of formally represented knowledge is based on a *conceptualization*: the objects, concepts, and other entities that are presumed to exist in some area of interest and the relationships that hold among them [Genesereth & Nilsson, 1987]. A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose. Every knowledge base, knowledge-based system, or knowledge-level agent is committed to some conceptualization, explicitly or implicitly.

An *ontology* is an explicit specification of a conceptualization. The term is borrowed from philosophy, where an ontology is a systematic account of Existence. For knowledge-based systems, what "exists" is that which can be represented. When the knowledge of a domain is represented in a declarative formalism, the set of objects that

can be represented is called the universe of discourse. This set of objects, and the describable relationships among them, are reflected in the representational *vocabulary*. Thus, the conceptualization can be specified by defining this vocabulary. For that reason, the word ontology has been associated in the AI literature with terminology, often organized into taxonomic hierarchies. In the context of formal knowledge representation, we say that an ontology is a *set of definitions* of representational vocabulary—the vocabulary with which a knowledge-based program represents knowledge about things in its universe of discourse. By *definition* we mean the name of some entity in the universe of discourse (e.g., a class, relation, function, or object), human-readable text describing what the name is meant to denote, and declarative constraints (axioms) on the well-formed use of the name.

Current research is exploring the idea of using ontologies to specify interfaces for reusable knowledge-system modules and protocols for knowledge-level communication among cooperating software agents [Cutkosky et al., 1992; Genesereth, 1992; Gruber, Tenenbaum, & Weber, 1992; Neches et al., 1991; Patil et al., 1992; Walther, Eriksson, & Musen, 1992]. A long-term objective of such work is to enable libraries of reusable knowledge components and knowledge-based services that can be invoked over networks.

This paper addresses the problem of portability for ontologies. Portability is a problem because the parties to a common ontology may use different representation languages and systems. Ideally, shared terms should be defined at the knowledge level, independent of specific representation languages. Of course, definitions need to be couched in *some* common formalism if they are to be shareable by knowledge-based applications. However, it is not realistic or desirable to require that those applications be *implemented* in a common representation language or system. This is because different applications require different kinds of reasoning services, and special-purpose languages to support them. Thus, the portability problem for ontologies is to support common ontologies over multiple representation systems.

We describe a translation approach to the portability problem for ontologies. In a translation approach, ontologies are specified in a standard, system-independent form and translated into specific representation languages. In Section 3 we will describe Ontolingua, an implemented system for translating ontologies from a declarative, predicate-calculus language into a variety of representation systems. But first, we describe the motivation for explicit specifications of shared vocabulary.

2. Ontologies and knowledge sharing

Knowledge-based systems and services are expensive to build, test, and maintain. A software engineering methodology based on formal specifications of shared resources, reusable components, and standard services is needed. We believe that specifications of shared vocabulary can play an important role in such a methodology.

Several technical problems stand in the way of shared, reusable knowledge-based software. Like conventional applications, knowledge-based systems run on heterogeneous hardware platforms, are written in various programming languages, and

communicate using multiple network protocols. Emerging open-systems technology should provide the means for connecting programs over networks and across platforms. However, knowledge-based software poses additional requirements for interoperability. Such programs operate on and communicate using statements in a formal knowledge representation. They ask queries and give answers. They take "background knowledge" as an input. And as agents in a distributed AI environment, programs need a formalism in which to negotiate and exchange knowledge. For such knowledge-level communication, there is a need to establish conventions at the levels of representation language, agent communication protocols, and the content of shared knowledge [Patil et al., 1992]. Proposals for standard knowledge representation formats [Fulton, 1991; Genesereth & Fikes, 1992; Morik, Causse, & Boswell, 1991] and agent communication languages [Finin, Fritzson, & McKay, 1992] are independent of the content of knowledge being exchanged or communicated. Ontologies can be used to specify content-specific agreements.

Consider the problem of reusing a knowledge-based planning program. Such a program takes descriptions of objects, events, resources, and constraints, and produces plans that assign resources and times to objects and events. Although it may use general planning algorithms, like all knowledge-based systems the planner depends on a custom knowledge base (sometimes called a "domain theory" or "background knowledge") to get the job done. The knowledge base may contain some knowledge generic to the planning task, and some that describes the domain situations in which the planner is to run.

If one wished to use the planning system, one would need to adapt an existing knowledge base to a new application domain, or build one from scratch. This requires, at a minimum, a formalism that enables a human user to represent the knowledge so that the program can apply it. Furthermore, the developer needs to know the kinds of information given in inputs and returned as outputs, and the kinds of domain knowledge that is needed by the planner to perform its task. If the planning program were offered as a service that could be invoked over the network, or if a large planning problem were subcontracted out to several cooperating planning agents, then one would need an agreement about the topics of conversation that agents are expected to understand.

Underlying these content-specific agreements are *ontological commitments:* agreements about the objects and relations being talked about among agents, at software module interfaces, or in knowledge bases. For instance, developers and users of the planning system might agree on the definitions of objects that "provide," "require," and "produce" resources; time that comes in "points" and "durations" (including infinitely long and short durations); resources that are "allocated," "deallocated," and "produced" in "events"; and plans consisting of "state descriptions", "preferences" over states, and "effects" prescribed by "causal rules", events, or "operators."[1]

Ideally, we would like to be able to specify ontological commitments at the knowledge level [Newell, 1982]. We say that an agent commits to a knowledge-level specification if

[1]The terms is this example are taken from a common ontology defined for the DARPA/Rome Planning and Scheduling Initiative [Allen & Lehrer, 1992].

its observable actions are logically consistent with the specification.[2] This means that the specification format is independent of the internal (symbol-level) representation of the agent. With a bit of anthropomorphic license, the knowledge-level perspective can be applied to programs and knowledge bases. The "actions" of programs are "observed" through inputs and outputs, and specifications about their behavior may be given in terms of the formal arguments of a functional interface. Similarly, knowledge bases can be observed through a *tell and ask* functional interface [Levesque, 1984], where a client interacts with a knowledge base by making logical assertions (tell) posing queries (ask). The agents doing the telling and asking are viewed as black boxes.

In the context of multiple agents (including programs and knowledge bases), a *common ontology* can serve as a knowledge-level specification of the ontological commitments of a set of participating agents. A common ontology defines the vocabulary with which queries and assertions are exchanged among agents. For example, the words in the planning ontology are technical terms that govern the form of inputs and the interpretation of outputs. The definitions tell the user of a planning system what information must be given about an "event" or "resource" in order for the planner to be able to use the information. Each program must commit to the semantics of the terms in the common ontology, including arbitrary axioms about the properties of objects and how they are related.[3] However, there need be no commitment to the form or content of knowledge *internal* to the agent.

Furthermore, the axiomatization in an ontology need not be a complete functional specification of the behavior of an agent. Common ontologies typically specify only some of the formal constraints that hold over objects in the input and output in the domain of discourse of a set of agents.

Definitions in common ontologies are like the global type declarations in a conventional software library, and the ontological commitments are specified with type restrictions over the inputs and outputs of program modules. Formal argument restrictions can be checked mechanically by compilers, to help ensure that calling procedures pass legal data to called procedures. Similarly, sentences in a tell-and-ask exchange can be checked for logical consistency with the definitions in ontologies. Just as the formal argument list hides the internal workings of a procedure from its environment, a common ontology allows one to interact with a knowledge-based program without committing to its internal encoding of knowledge.

Ontologies are also like conceptual schemata in database systems. A conceptual schema provides a logical description of shared data, allowing application programs and databases to interoperate without having to share data structures. While a conceptual schema defines relations on *Data*, an ontology defines terms which with to represent *Knowledge*. For present purposes, one can think of Data as that expressible in ground atomic facts and Knowledge as that expressible in logical sentences with existentially and

---

[2]The view of software specifications as commitments is described by Mark et. al. [Mark, Tyler, McGuire, & Schlossberg, 1992].

[3]Of course, it is the programmers of those systems who commit to using terms consistently, and "share the meanings" of the terms.

universally quantified variables. An ontology defines the vocabulary used to compose complex expressions such as those used to describe resource constraints in a planning problem. From a finite, well-defined vocabulary one can compose a large number of coherent sentences. That is one reason why vocabulary, rather than form, is the focus of specifications of ontological commitments.

There are many aspects of the knowledge sharing problem that are not addressed by common ontologies. Questions not addressed include how groups of people can reach consensus on common conceptualizations, and how terms can be defined outside their context of use. The utility of common ontologies as a sharing mechanism is a hypothesis, the subject of collaborative studies [Patil et al., 1992]. Ontolingua is a tool to support such research.

3. Ontolingua: A system for portable ontologies

Ontolingua is a system for describing ontologies in a form that is compatible with multiple representation languages. It provides forms for defining classes, relations, functions, objects, and theories. It translates definitions written in a standard, declarative language into the forms that are input to a variety of implemented representation systems. Ontologies written in Ontolingua can thereby be shared by multiple users and research groups using their own favorite representation systems, and can be ported from system to system. Figure 1 shows some of the possible applications of a single ontology translated into several systems.

The syntax and semantics of Ontolingua definitions are based on a notation and semantics for an extended version of first-order predicate calculus called Knowledge Interchange Format (KIF) [Genesereth & Fikes, 1992]. KIF is intended as a language for the publication and communication of knowledge. It is intended to make the *epistemological-level* [McCarthy & Hayes, 1969] content of a knowledge base clear to the reader, but *not* to support automated reasoning in that form. It is very expressive, designed to accommodate the state of the art in knowledge representation. But it is not an implemented representation system.

Ontolingua translates definitions written in KIF into forms appropriate to implemented representation systems, which typically impose a restricted syntax and support limited reasoning over a restricted subset of full first order logic. Currently, Ontolingua can translate into LOOM [MacGregor, 1991] (a KL-ONE style system), Epikit [Genesereth, 1990] (a predicate calculus language descended from MRS), Algernon[4] [Crawford & Kuipers, 1989] (a frame system based on Access Limited Logic), and pure KIF (i.e., it puts ontologies into a canonical form of KIF to facilitate further translation into other languages). It would be relatively simple to write translators into CycL [Lenat & Guha, 1990], KEE [Fikes & Kehler, 1985], or EXPRESS [Spiby, 1991]. Ontolingua can parse arbitrary KIF sentences, and recognizes many common representation idioms. However, not all KIF sentences can be translated into all target representation systems. When

[4]The Ontolingua-to-Algernon translator was implemented by Juan Carlos Martinez at MCC.

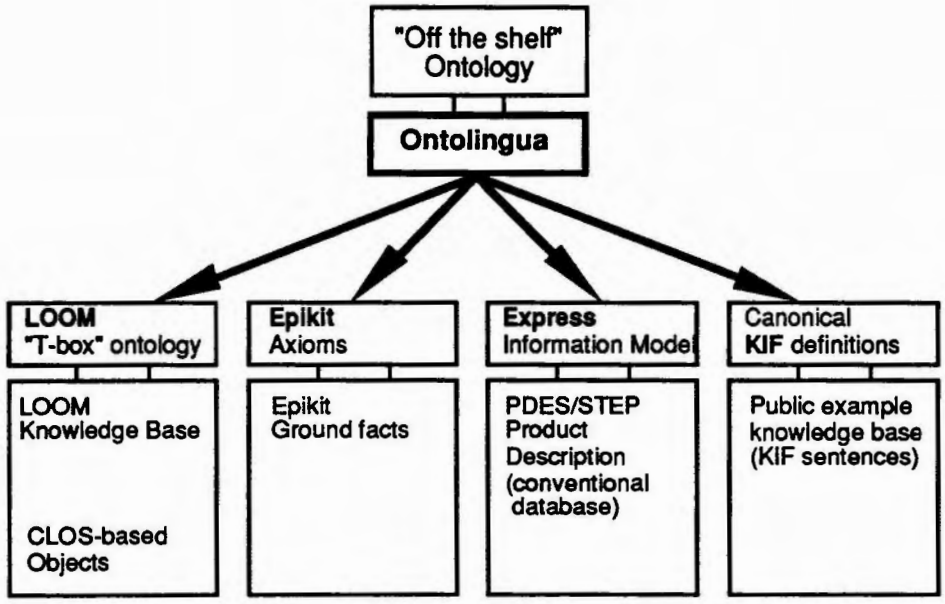


Figure 1: Translating from a common ontology into several applications. The same ontology can be used for different purposes in different systems. LOOM can be used during the conceptual design of the ontology, helping with classification. It can also be used to manage a knowledge base of facts about objects. Epikit is useful for exploratory reasoning; it provides a suite of powerful deductive engines. Express is the standard language for describing PDES information models, which are logical database designs for sharable product description data. Other public-domain knowledge bases, such as mode libraries and experimental data sets, can be shared via a common ontology in a canonical form of KIF definitions.

definitions contain sentences that cannot be translated into a given implementation, Ontolingua informs the user and continues.

The set of idioms that Ontolingua can recognize and translate is defined in an ontology, called the frame ontology. The frame ontology specifies, in a declarative form, the representation primitives that are often supported with special-purpose syntax and code in object-centered representation systems (e.g., classes, instances, slot constraints, etc.). We discuss the frame ontology in Section 3.6.

3.1 A brief introduction to the syntax and semantics

Ontolingua definitions are written in natural language and KIF sentences. The natural language text is not parsed or translated, but is passed on to those systems that have a place for documentation (surprisingly, many representation systems do not!). The technical task is to translate declarative sentences, which are axioms that constrain the meaning of the defined terms.

KIF provides a Lisp-like notation for writing the axioms in Ontolingua definitions. Objects are denoted by object constants (Lisp atoms), or term expressions, which are constructed from lists whose first element is a function constant. Sentences are formed from lists whose first element is a relation constant and remaining elements are terms. Individual variables that may be universally or existentially quantified are marked with

the '?' prefix. There are operators for conjunction, disjunction, implication, and negation. For example, the following is a KIF sentence that says, "All writers are misunderstood by some reader."

```
(forall ?W
  (=> (writer ?W)
    (exists ?R
      (and (reads ?R ?W)
        (not (understands ?R ?W))))
```

The symbol `writer` is a relation constant, and the sentence `(writer ?W)` says that `?W` is a writer. `?W` is a universally quantified variable, and `?R` is existentially quantified. `Reads` is a binary relation, and `(reads ?R ?W)` says that `?R` reads something by `?W`.

The KIF specification [Genesereth & Fikes, 1992] provides model-theoretic semantics for the language and an axiomatization of the primitive object types such as sets, lists, relations, and functions. Sets are defined using a standard set theory. Lists are finite sequences of objects. There is no distinction among linked lists, arrays, or any other encoding of sequences, since KIF is a declarative specification language and not an implementation. Relations are defined as sets of tuples, where each tuple is a list of objects. Functions are a special case of relations, in which the last object in each tuple is unique given the preceding objects. A function of N arguments is a relation of N+1 arguments in which the value of the last argument is a function of the first N arguments.

Ontolingua definitions are Lisp-style forms that associate a symbol with an argument list, a documentation string, and a set of KIF sentences labeled by keywords. An Ontolingua ontology is made up of definitions of classes, relations, functions, distinguished objects, and axioms that relate these terms.

A relation is defined with a form like the following:

```
(define-relation connects (?comp1 ?comp2)
  "The most general binary connection relation.
Two components can be connected if neither is a subpart-of the other."
  :def (and (component ?comp1)
        (component ?comp2)
        (not (subpart-of ?comp1 ?comp2))
        (not (subpart-of ?comp2 ?comp1)))
```

The arguments `?comp1` and `?comp2` are universally quantified variables ranging over the items in the tuples of the relation. This example is a binary relation, so each tuple in the relation has two items. Relations of greater arity can also be defined. The sentence after the `:def` keyword is a KIF sentence stating logical constraints over the arguments. Constraints on the value of the first argument of a binary relation are effectively domain restrictions, and those on the second argument of a binary relation are range restrictions. There may also be complex expressions stating relationships among the arguments to a relation. The `:def` constraints are *necessary* conditions, which must hold if the relation holds over some arguments. It is also possible to state sufficient conditions, or any combination. In textbook notation for predicate calculus, the form above means:

$$\forall x,y \ \text{connects}(x,y) \Rightarrow \text{component}(x) \wedge \text{component}(y) \wedge \neg\text{subpart-of}(x,y) \wedge \neg\text{subpart-of}(y,x).$$

If the `:iff-def` keyword is used instead of `:def`, the definition becomes necessary *and* sufficient. That is:

$$\forall x,y \ \text{connects}(x,y) \Leftrightarrow \text{component}(x) \wedge \text{component}(y) \wedge \neg\text{subpart-of}(x,y) \wedge \neg\text{subpart-of}(y,x).$$

A class is defined with a similar form, where there is exactly one argument, called the instance variable. In Ontolingua, classes are treated as unary relations to help unify object- and relation-centered representation styles.

```
(define-class connection (?connection)
  "A connection is a special case of module where there are no
internal variables. They can be thought of as conduits for transfer
of energy/matter/information among modules. They also can be viewed
as an encapsulation of intermodule constraints. Connections are typed
by the number and type of their ports."
  :def (and (module ?connection)
            (not (exists ?variable
                   (internal-state-variables ?connection ?variable)))))
```

The `:def` sentence describes those things that must be true of instances of the class, that is, necessary constraints. Unary relations applied to the instance variable specify superclasses of the class being defined. For example, `connection` is a subclass of `module` by definition, because every connection `?connection` must also be a `module`. Other properties of instances can be stated using KIF sentences mentioning the instance variable. The logical meaning of the class definition above is

$$\forall x \ [\text{connection}(x) \Rightarrow \text{module}(x) \wedge \neg\exists y \ \text{internal-state-variables}(x,y)\ ]$$

A function is defined like a relation, with a slight variation in syntax, moving the final argument outside of the argument list:

```
(define-function squared (?n) :-> ?value
 "The squared of a number is the product of it times itself."
  :def (and (number ?n)
            (number ?value))
  :lambda-body (* ?n ?n))
```

As in definitions of relations, the arguments to a function are constrained with necessary conditions following the `:def` keyword. The function is only defined on arguments that satisfy these domain constraints (i.e., `?n` is a number). A restriction on the range of the function is specified with a sentence constraining the value variable after the `:->` keyword (i.e., `?value` is a number). The body of the `define-function` form is the same as the `define-relation` form, except it also allows a `:lambda-body` to be given. A lambda-body is a KIF term expression that denotes the value of the function in terms of its arguments. In the example, * is a function and the lambda-body is a term that denotes the product of `?n` and itself. Since definitions are purely declarative, the

functions in a lambda body are "side-effect free." The logical meaning of the example function definition is

$$\forall x, y\,[y = \text{squared}(x)] \Rightarrow \text{number}(x) \wedge \text{number}(y),$$
$$\forall x\ \text{squared}(x) = x \cdot x$$

Functions need not be defined on all objects. When a function is applied to argument on which it is undefined, the term expression denotes a special KIF object called `bottom` ($\perp$). Thus, another way to describe the example function is

$$\text{squared}(x) = \begin{cases} x^2 & \textit{if}\ \text{number}(x) \\ \perp & \textit{otherwise} \end{cases}$$

Finally, it is possible to define distinguished individuals in an ontology. Although Ontolingua is intended for defining vocabulary, rather than storing the ground facts of a knowledge base, sometimes the name of a particular constant is part of the vocabulary. For example:

```
(define-instance zero (number)
  "Zero is the identity element for addition."
  :axiom-def (forall ?x
          (=> (number ?x)
            (and (= (+ ?x zero) ?x) (= (+ zero ?x) ?x)))
  := 0)
```

This form defines the object constant `zero`. The list `(number)` says that `zero` is an instance of the class `number`. An object can be an instance of more than one class. The sentence labeled with `:axiom-def` is a stand alone axiom constraining the object constant. After the `:=` keyword is a KIF term expression that denotes the object being defined. It is like the `:lambda-body` expression for functions, but without free variables.

Details on the language and software are documented in a longer report [Gruber, 1992] and documentation distributed with the software.

3.2 An example definition

Consider an ontology for bibliographic information. The purpose of a bibliography ontology is to support knowledge sharing tasks such as exchanging bibliographic data among databases, integrating bibliographic databases with other databases (e.g., address books, company directories), linking citations across hyperdocument boundaries, and providing network-based services for bibliographic data processing. The domain of discourse of these tasks includes documents, authors, publishers, dates, places, and the references that occur in documents and card catalogs. In this ontology, authors are defined as follows.[5]

---

[5]The relations `value-cardinality` and `value-type` are defined in the frame ontology, which is included with Ontolingua (see the appendix). The relations `author.name`, `biblio-name`, `author.documents`, and `person.name` are defined in the bibliography ontology. Operators such as `and`, relations such as `=>`, `=`, `<=>`, and numbers such as `1` are defined by the KIF specification.

```
(define-class AUTHOR (?author)
  "An author is a person who writes things.
An author must have created at least one document.
In this ontology, an author is known by his or her real name."
  :def (and (person ?author)
        (= (value-cardinality ?author author.name) 1)
        (value-type ?author author.name biblio-name)
        (>= (value-cardinality ?author author.documents) 1)
        (<=> (author.name ?author ?name)
           (person.name ?author ?name))))
```

This form defines the term author, which denotes a class. Individual authors such as Marvin Minsky are instances of the class. The `:def` sentence gives necessary conditions on class membership. The first conjunct of the sentence, `(person ?author)`, says that all authors must also be persons. The second conjunct says that authors must have exactly one associated name, given by the relation `author.name`. This means that the relation `author.name` maps every instance of the class `author` to some name, and that the name is a function of the author.[6] The third conjunct specifies the type restriction that the value of the `author.name` relation, when applied to instances of `author`, must be an instance of the class `biblio-name`. The fourth conjunct says that there must be some document(s) associated with every author by the relation `author.documents`. The fourth conjunct specifies that the `author.name` and the `person.name` of an author are the same. This constraint explicitly rules out calling authors by pen names.

3.3 The ontological commitments implied by definitions

What is the nature of the "ontological commitments" specified by such definitions? Agents that commit to the ontology agree to use the terminology in a manner that is consistent with the axioms and documentation strings. One can take advantage of such a commitment when using knowledge services. For instance, imagine that a program asks for the author of some document, perhaps with this query:

```
(ask ?x (author.documents ?x society-of-mind))
```

The database service might return the identifier `author-423455` as a binding for `?x`. Then the asking program could request the name of the author with this query:

```
(ask ?name (author.name author-423455 ?name))
```

From the definition of `author`, the asking program can now count on the name returned by the `author.name` query to be the same as the name of the *person* Marvin Minsky. It could then use that name when querying other knowledge bases about famous people, only some of whom are authors.

An agent that commits to the bibliography ontology does not necessarily have to *store* a database of authors, or to be able to answer any query about authors. The ontological

---

[6] If `author.name` were defined separately as a function, then the definition would say that it maps each element of its domain to at most one name. The `has-one-of-type` statement does not define the relation; it only restricts its value when applied to instances of the class `author`. This minimal commitment is consistent because all functions are also relations.

commitments are still useful under partial knowledge. Consider the case where there are several bibliography knowledge servers on the network that are committed to the bibliography ontology, each with incomplete knowledge. Imagine we ask one knowledge server for the books by Marvin Minsky, and it replies that it isn't aware of any. That does not mean that there are no such books, of course. We can assume that there must be some works written by Minsky, since he was called an author. We can then query other knowledge servers, until we find one that knows the answer. Similarly, if we asked for the name of an author in a distributed fashion and got an answer, we know from the definition that there is at most one name per author, so we can stop asking other knowledge servers for his name.

3.4 Translation into representation systems

The definition of the author class specifies constraints that are typical of descriptions in many systems. Let us look at how it translates into different representation languages.

In Epikit, a predicate calculus language, the author definition translates to

```
(=> (author $author)
   (person $author))
(=> (author $author)
   (exists $y (author.name $author $y) (biblio-name $y)))
(=> (author $author)
   (=> (author.name $author $y) (author.name $author $y2)
      (= $y $y2)))
(=> (author $author)
   (exists $y (author.documents $author $y)))
(=> (author $author)
   (<=> (author.name $author $name)
      (person.name $author $name)))
```

Epikit is based on an early version of KIF; it is a prefix-form predicate calculus with equality and functional term expressions. Epikit does not distinguish definitions from any other axioms. Thus, the necessary conditions of the Ontolingua definition get turned into implications. Epikit also does not support reasoning about relations as objects. In the Ontolingua definition some of the relations take classes and other relations as arguments. For example, the relation `value-type` took the binary-relation `author.name` and the class `biblio-name` as arguments. Relations that take other relations as arguments are called *second-order relations* (we will say more about them in Section 3.6). To translate such statements into pure first-order systems such as Epikit, Ontolingua recognizes the second-order relations and instantiates axiom schemata for them. For example,

```
(>= (value-cardinality ?author author.documents) 1)
```

becomes

```
(exists $y (author.documents $author $y)).
```

In LOOM, a object-oriented KL-ONE-style language, the author definition translates as

```
(define-concept author
 :is (:and :primitive
        person
        (:the author.name biblio-name)
        (:at-least 1 author.documents)
        (:same-as author.name person.name)))
```

The LOOM translation looks similar to the Ontolingua form, except that there are no free variables. The second-order relations such as `value-type` and `value-cardinality` were fashioned after the analogous operators in LOOM derivatives such as `:all` and `:at-least`. They capture common constraints that hold between classes and binary relations applied to their instances. However, in Ontolingua, these second-order relations are not part of the syntax or primitive operators of the language. They are just vocabulary—relations defined in KIF—for specifying portable ontologies and translating among LOOM-like systems.

In KEE, an object-oriented, frame-based system, the example would appear like this in a frame browser:

```
Unit: AUTHOR
Comment: "An author is a person who writes things.
    An author must have created at least one document.
    In this ontology, an author is known by his or her real name."
Superclasses: PERSON
Member Of: CLASSES
Member slot: AUTHOR.NAME
 ValueClass: (MEMBER.OF BIBLIO-NAME)
 Min.Cardinality: 1
 Max.Cardinality: 1
Member slot: AUTHOR.DOCUMENTS
 Min.Cardinality: 1
Member slot: PERSON.NAME from Person   ;;can't translate the constraint
```

KEE is a distant cousin of LOOM, and it also has special syntax for slot constraints (e.g., Valueclass, Min.Cardinality). However, the KEE frame system is less expressive than LOOM and the logic-based languages. Consequently, a constraint that would be said declaratively in LOOM might be implemented with an attached procedure in KEE. For instance, the KEE frame browser shows a template for the slot `person.name` "inherited" from the superclass `person`, but it does not have a constraint language capable of expressing the fact that the value of the `author.name` slot is the same as the value of the `person.name` slot. To implement this constraint, one would write an attached procedure that computes the value of one of the slots from the other if either is given a value. The existence of embedded procedural code in object-oriented systems such as KEE is one reason for maintaining ontologies in a more expressive, declarative language, and translating *into* the restricted languages. The assumption is that it is easier to generate procedural code to implement a declarative specification than to infer the declarative semantics from a procedure.

3.5 Translation architecture

The design of Ontolingua was constrained by three competing requirements:

- to offer an expressive, declarative, system-independent language in which to write definitions
- to support translation into specialized representations with restricted expressive and inferential power
- to allow for easy extension in expressiveness and target implementations

It is impossible achieve all three of the requirements completely. If the source language is more expressive than the target languages, then the translation will be incomplete. If one built separate, special-purpose translators for each language, the subset of KIF that is handled would depend on the target language. And each additional translator would be as expensive to implement as the first.

Ontolingua's design strikes a balance in which translation is complete only for a restricted subset of representational idioms, but these are defined independently of any target system. For each of these idioms, Ontolingua code *recognizes* its expression in KIF and transforms it into a *canonical form*. The canonical form is an intermediate representation that facilitates translation into multiple target languages. Sentences in the canonical form have a predictable format that simplifies pattern matching by reducing the number of ways that an equivalent thing can be stated.

For example, in the `author` example, the necessary condition

```
:def (... (person ?author) ...)
```

is an idiom that is transformed by Ontolingua into the stand alone axiom

```
(subclass-of author person)
```

Similarly, the slot equivalence constraint

```
(<=> (author.name ?author ?name)
   (person.name ?author ?name))
```

is turned into

```
(have-same-slot-values author author.name person.name).
```

In both of these cases, Ontolingua recognized a representational idiom, and transformed it into a logically equivalent sentence using a *second-order* relation. A second-order relation is a relation that can take other relations (including classes and functions) as arguments. For example, the statement `(range` *relation class*) specifies that all values of the given binary *relation* must be an instance of *class*. `Range` is a second-order relation that is recognized by Ontolingua. This constraint could be expressed in several equivalent ways in KIF:

```
(=> (relation ?x ?y) (class ?y))
(forall (?x ?y) (=> (relation ?x ?y) (class ?y)))
(<= (class ?y) (relation ?x ?y))
(=> (member ?tuple relation) (class (second ?tuple)))
...
```

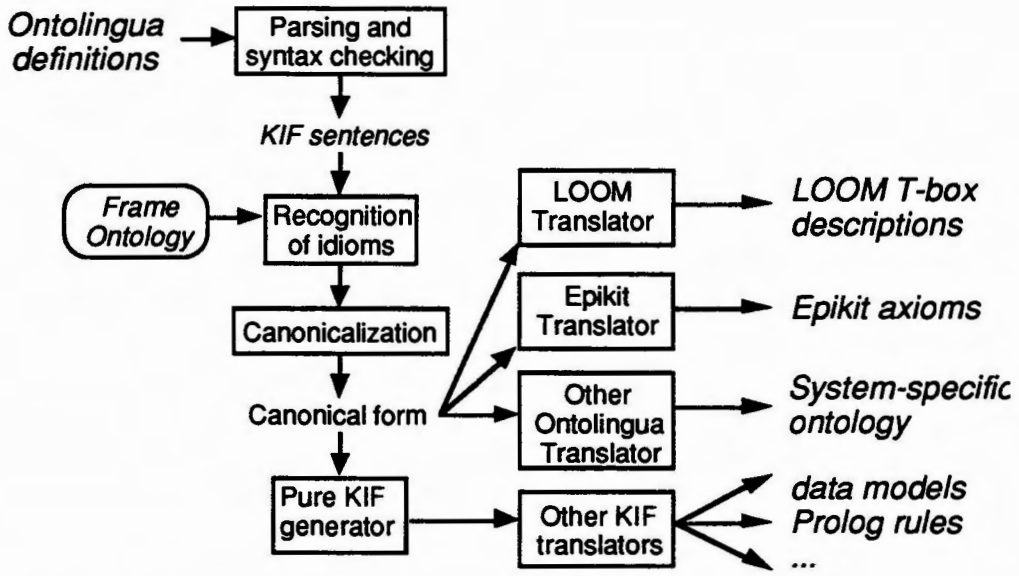


Figure 2: Translation architecture for Ontolingua. Instances of common representation idioms are recognized and transformed into a simpler, equivalent form using the second-order vocabulary from the Frame Ontology. These and other transformations result in a canonical form, which is given to back-end translators that generate system-specific output. A pure KIF output is also available at this stage, to be given to other translators developed for KIF, such as KIF-to-Prolog.

Recognizing idioms and transforming them into canonical form are two of the front-end processes that Ontolingua performs when translating. Figure 2 shows the data flow of the translation architecture. Once definitions have been put into canonical form, back-end translators transform them into output appropriate to specific target systems. This architecture is analogous to a conventional programming language compiler, which parses source into an intermediate form that is then given to specialized code generation modules. The system-independent translation work is done in the first phase, simplifying the work for the system-specific translators. Since the front-end Ontolingua processor can recognize and transform the common variants, the back-end translators have a finite set of "special cases" to consider.

3.6 The Frame Ontology

The choice of supported second-order relations was made to capture common knowledge-organization conventions used in object-centered or frame-based representations. The second-order relations recognized by Ontolingua are defined in an ontology, called the *Frame Ontology*. It contains a complete axiomatization of classes and instances, slots and slot constraints, class and relation specialization, relation inverses, relation composition, and class partitions. Each second-order term is defined in natural language and KIF axioms. A current list of the Frame Ontology vocabulary is given in the appendix. By no coincidence, these are the constructs that are supported with special-

purpose syntax and code in object-centered representation systems. Together, they constitute a purely declarative representational framework for describing hierarchies of classes with slots. Users of Ontolingua and developers of translators can consequently share a well-defined vocabulary and make assumptions about its consistent use.

The frame ontology is carefully designed to allow both relational and object-centered styles of representation to co-exist (and it is consistent with the built-in ontologies in KIF for relations and functions). The design choices are not arbitrary. There are several possible ways to axiomatize classes, for example. Some systems treat them as simple sets; others as unary relations; others as intentional entities or descriptions. The frame ontology defines classes to be coextensional with unary relations, so that the relational-style `(person Fred)` is compatible with the object-style `(instance-of Fred person)`. Similarly, slots can be viewed as binary relations, unary functions onto sets, projections of relations, or some combination. The frame ontology makes a minimal commitment that allows slots to be viewed as binary relations or unary functions, and does not commit to distinctions among attributes, properties, and slots as local to classes (see [Guarino, 1992; Sowa, 1992]).

We have already described how the second-order vocabulary can facilitate translation by providing a canonical form for representation idioms. The frame ontology vocabulary can also be used by ontology writers directly in their definitions, as a succinct representation for simple things that can be awkward in pure first-order predicate calculus. For example, in the `author` definition the relation `value-cardinality` was used as follows

```
(= (value-cardinality ?author author.name) 1)
```

which could have been stated using this form:

```
(and (exists ?name (author.name ?author ?name))
   (forall (?name1 ?name2)
      (=> (and (author.name ?author ?name1)
               (author.name ?author ?name2))
          (= ?name1 ?name2))))
```

The same second-order relation is also used in this constraint:

```
(>= (value-cardinality ?author author.documents) 1)
```

which might otherwise have been stated using this idiom:

```
(exists ?document (author.documents ?author ?document)).
```

Note that second-order relations are not substitutional macros for the idioms. Obviously the idioms for these two applications of value cardinality are not expanded from the same template. Compare them to the definition of `value-cardinality` as given in the Frame Ontology, which uses a set-theoretic axiomatization instead of existential quantification.

```
(define-function VALUE-CARDINALITY (?instance ?binary-relation) :-> ?n
  "The VALUE-CARDINALITY of a binary-relation with respect to a given
domain instance is the number of range-elements to which the relation
maps the domain-element. For a function (single-valued relation), the
VALUE-CARDINALITY is 1 on all domain instances for which the function
is defined. It is 0 for those instances outside the exact domain of
the function."
  :lambda-body (cardinality (setofall ?y
                              (holds ?binary-relation ?instance ?y)))
```

Using the second-order relations of the Frame Ontology, one can also describe properties of classes, functions, and other relations, without mentioning the instance arguments or instance variables. For example, the following defines the class of things that can be the name of authors:

```
(define-class author-name (?name)
  "Names that belong to authors."
  :axiom-def (exact-range author.name author-name))
```

the `:axiom-def` label indicates that this is a stand alone axiom that defines the class. The same constraint could have been specified in a necessary-and-sufficient condition using the free variable `?name`:

```
(define-class author-name (?name)
  "Names that belong to authors."
  :def (exists ?author (author.name ?author ?name)))
```

Independent of Ontolingua, the frame ontology can serve as a framework for defining representational clichés and idioms. The same conventions facilitate translation *into* KIF from object-centered systems, offering a canonical form for stating things that might be written in several equivalent forms. With feedback from Ontolingua and the knowledge representation community, we expect to add new second-order relations to accommodate the expressive needs for defining common ontologies. Fortunately, the cost of adding a new second-order relation is limited to the effort required to support that "special case" in each target representation. Importantly, the syntax, operators, and semantics of the shared language need not change.

4. Discussion

Ontolingua is not a replacement for a representation system like CycL, Epikit, or LOOM. One of the motivations for trying to make ontologies portable over several implementations of representation systems, rather than just standardizing on one system, is that different systems provide different computational services at different costs. The translation strategy allows one to use some computational services at conceptual design time (e.g., terminological classification, general-purpose inference) and to use the same ontology for the development of production systems (e.g., translating it into object-oriented class definitions or database schemata). It is important to note that Ontolingua does not support query processing; users call implemented systems directly to invoke inference procedures. This lessens the need to write ontologies in a special-purpose

language, because many of the compromises made in the expressiveness of a representation are made to optimize run-time inference.

Because it translates into restricted languages, Ontolingua is inherently incomplete with respect to the KIF language. The only way to be truly portable over the specialized systems is to take a common-denominator approach, writing definitions using only those constructs known to be supported in all of the systems of interest. The frame ontology is a compromise: it identifies a set of *some* common idioms that are supported in *most* of the target systems. Although it can parse any legal KIF expression, Ontolingua is only designed to translate a subset of KIF (which is described in the documentation). For instance, the current implementation is not guaranteed to translate sentences using some of the more sophisticated language features of KIF, such as meta-level operators. It currently does not support user-defined second-order relations; only the terms defined in the frame ontology are recognized.

However, the full, first-order predicate calculus is available to the ontology writer. Consequently, incompleteness is a fact of life in the representation translation business. When Ontolingua cannot translate a sentence into a target implementation, it issues an informative message. The practical consequence of not translating a sentence could be that the target system may be unable to enforce a constraint, or it may have to fall back on inefficient theorem proving. The good news is that target systems can be customized or replaced without changing the ontology.

Ontolingua is a domain-independent translation tool; it does not help with the intellectual task of designing ontologies. Deciding which concepts and relations to include, and writing axiomatic definitions, requires knowledge that is not found in existing knowledge bases. Ontologies like the frame ontology and the axiomatization of set theory in KIF are called *representation ontologies*. Representation ontologies provide a framework, but do not offer guidance about how to represent the world. *Content ontologies* make claims about how the world (or a conceptualization of it) should be described. Some content ontologies are intended to be comprehensive—to be "conceptual coat rack" on which to "hang" more specific ontologies and domain knowledge. Examples of comprehensive content ontologies include Cyc's global ontology [Lenat & Guha, 1990] and the Penman Upper Model [Bateman, Kasper, Moore, & Whitney, 1990]. Whether these ontologies can help in the design of more specialized ontologies is an empirical question. Ontolingua was created to support experimentation by making such ontologies accessible as off-the-shelf artifacts.

As a representation ontology, the frame ontology *does* specify a conceptualization implicit in knowledge bases written in the object-centered style. Classes, specialization, and slot constraints are not built into KIF; they are a convention supported in specialized representation-system architectures. The frame ontology defines these object-centered concepts in a declarative form that is stylistically compatible with pure relation-centered usage. The vocabulary for these concepts defines what can be translated, and offers a succinct form for users to write portable ontologies. In these ways, the frame ontology is a specification of the ontological commitments of Ontolingua.

Acknowledgments

This work was motivated by collaborative activities of the DARPA Knowledge Sharing Effort. Thanks for insights and advice from Richard Fikes, Mike Genesereth, Bill Mark, Bob MacGregor, Bob Neches, Ramish Patil, and Marty Tenenbaum. Karl "Fritz" Mueller redesigned and implemented the current version of the software. Special thanks to Richard Fikes, Hania Gajewska, and Charles Petrie for thoughtful reviews. Funding for this work is provided by DARPA prime contract DAAA15-91-C-0104 through Lockheed subcontract SQ70A3030R, and NASA Grants NCC 2-537 and NAG 2-581 (under ARPA Order 6822).

References

Allen, J. & Lehrer, N. (1992). DARPA/Rome Laboratory Planning and Scheduling Initiative Knowledge Representation Specification Language Version 2.0. ISX, Unpublished report

Bateman, J. A., Kasper, R. T., Moore, J. D., & Whitney, R. A. (1990). A General Organization of Knowledge for Natural Language Processing: The Penman Upper Model. USC/Information Sciences Institute, Marina del Rey, CA, Unpublished research report

Crawford, J. M. & Kuipers, B. J. (1989). Toward a theory of access-limited logic for knowledge representation. *Proceedings of the First International Conference on Principles of Knowledge Representation*, Morgan Kaufmann.

Cutkosky, M., Engelmore, R., Fikes, R., Gruber, T., Genesereth, M., Mark, W., Tenenbaum, J., & Weber, J. (1992). PACT: An experiment in integrating concurrent engineering systems. *IEEE Computer*, . Submitted to special issue on computer-supported concurrent engineering.

Fikes, R. & Kehler, T. (1985). The role of frame-based representation in reasoning. *Communications of the ACM*, **28**(9):904-920.

Finin, T., Fritzson, R., & McKay, D. (1992). An Overview of KQML: A Knowledge Query and Manipulation Language. Department of Computer Science, University of Maryland, Technical Report

Fulton, J. A. (1991). The semantic unification meta-model: technical approach. IGES/PDES Organization, Dictionary/Methodology Committee, Standards working document ISO TC184/SC4/* WG3 N 81 (P 0). Contact James Fulton, Boeing Computer Services, P. O. Box 24346, MS 7L-64, Seattle, WA 98124-0346.

Genesereth, M. (1990). *The Epikit Manual.*

Genesereth, M. R. (1992). An Agent-Based Framework for Software Interoperability. *Proceedings of the DARPA Software Technology Conference,* Meridian Corporation, Arlington VA, pages 359-366. Also Report Logic-92-2, Computer Science Department, Stanford University, June 1992.

Genesereth, M. R. & Fikes, R. E. (1992). Knowledge Interchange Format, Version 3.0 Reference Manual. Computer Science Department, Stanford University, Technical Report Logic-92-1.

Genesereth, M. R. & Nilsson, N. J. (1987). *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers.

Gruber, T. R. (1992). Ontolingua: A mechanism to support portable ontologies. Stanford University, Knowledge Systems Laboratory, Technical Report KSL 91-66. Revision.

Gruber, T. R., Tenenbaum, J. M., & Weber, J. C. (1992). Toward a knowledge medium for collaborative product development. *Proceedings of the Second International Conference on Artificial Intelligence in Design,* Pittsburgh, pages 413-432, Kluwer Academic.

Guarino, N. (1992). Concepts, Attributes, and Arbitrary Relations: Some Linguistic and Ontological Criteria for Structuring Knowledge Bases. Italian National Research Council, To appear in *Data and Knowledge Engineering*, 8 (1992).

Lenat, D. B. & Guha, R. V. (1990). *Building Large Knowledge-based Systems: Representation and Inference in the Cyc Project.* Menlo Park, CA: Addison-Wesley.

Levesque, H. J. (1984). Foundations of a functional approach to knowledge representation. *Artificial Intelligence*, **23**:155-212.

MacGregor, R. (1991). The Evolving Technology of Classification-based Knowledge Representation Systems. In J. Sowa (Eds.), *Principles of Semantic Networks: Explorations in the Representation of Knowledge.* San Mateo, CA: Morgan Kaufmann.

Mark, W., Tyler, S., McGuire, J., & Schlossberg, J. (1992). Commitment-based software development. *IEEE Transactions on Software Engineering*, (Oct.).

McCarthy, J. & Hayes, P. J. (1969). Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer & D. Michie (Eds.), *Machine Intelligence 4.* Edinburgh: Edinburgh University Press.

Morik, K., Causse, K., & Boswell, R. (1991). A Common Knowledge Representation Integrating Learning Tools. GMD,

Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T., & Swartout, W. R. (1991). Enabling technology for knowledge sharing. *AI Magazine*, **12**(3):16-36.

Newell, A. (1982). The knowledge level. *Artificial Intelligence*, **18**(1):87-127.

Patil, R. S., Fikes, R. E., Patel-Schneider, P. F., McKay, D., Finin, T., Gruber, T. R., & Neches, R. (1992). The DARPA Knowledge Sharing Effort: Progress report. *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference,* Cambridge, MA, Morgan Kaufmann.

Sowa, J. F. (Eds.). (1992). *Principles of Semantic Networks.* San Mateo, CA: Morgan Kaufmann Publishers.

Spiby, P. (1991). EXPRESS Language Reference Manual. Project Three ("Express Language") Working Group Five ("STEP Development Methods") of Subcommittee Four ("Manufacturing Languages and Data") of ISO Technical Committee 184 ("Industrial Automation Systems and Integration"); International Standards Organization, International Standard ISO 10303-11.

Walther, E., Eriksson, H., & Musen, M. A. (1992). Plug and play: Construction of task-specific expert-system shells using sharable context ontologies. Knowledge Systems Laboratory, Stanford University, Technical Report KSl-92-40.

Appendix: Vocabulary defined in the Frame Ontology

This appendix lists the vocabulary of the frame ontology (Version 3). The complete axiomatization and documentation is included with the Ontolingua software.

class **relation** (?relation)
class **function** (?function)
class **class** (?class)
relation **instance-of** (?individual ?class)
relation **subclass-of** (?child-class ?parent-class)
relation **superclass-of** (?parent-class ?child-class)
relation **subrelation-of** (?child-relation ?parent-relation)
relation **direct-instance-of** (?individual ?class)
relation **direct-subclass-of** (?child-class ?parent-class)
function **arity** (?relation) :-> ?n
function **exact-domain** (?relation) :-> ?domain-relation
function **exact-range** (?relation) :-> ?class
class **n-ary-relation** (?relation)
class **unary-relation** (?relation)
class **binary-relation** (?relation)
class **unary-function** (?function)
relation **single-valued** (?binary-relation)
function **inverse** (?binary-relation) :-> ?relation
function **projection** (?relation ?column) :-> ?class
function **composition** (?relation-1 ?relation-2) :-> ?binary-relation
function **compose** (@binary-relations) :-> ?binary-relation
relation **alias** (?relation-1 ?relation-2)
relation **domain** (?relation ?class)
relation **domain-of** (?domain-class ?binary-relation)
relation **range** (?relation ?class)
relation **range-of** (?class ?relation)
relation **values** (?domain-instance ?binary-relation ?set-of-values)
function **all-values** (?domain-instance ?binary-relation) :-> ?set-of-values
relation **value-type** (?domain-instance ?binary-relation ?class)
function **value-cardinality** (?domain-instance ?binary-relation) :-> ?n
relation **same-values** (?domain-instance ?relation-1 ?relation-2)
relation **inherited-slot-values** (?domain-class ?binary-relation ?set-of-values)
function **all-inherited-slot-values** (?domain-class ?binary-relation) :-> ?set-of-values
relation **slot-value-type** (?domain-class ?binary-relation ?range-class)
function **slot-cardinality** (?domain-class ?binary-relation) :-> ?n
relation **minimum-slot-cardinality** (?domain-class ?binary-relation ?n)
relation **maximum-slot-cardinality** (?domain-class ?binary-relation ?n)
relation **single-valued-slot** (?domain-class ?binary-relation)
relation **same-slot-values** (?domain-class ?relation-1 ?relation-2)
class **class-partition** (?set-of-classes)
relation **subclass-partition** (?c ?class-partition)
relation **exhaustive-subclass-partition** (?c ?class-partition)
relation **asymmetric-relation** (?binary-relation)
relation **antisymmetric-relation** (?binary-relation)
relation **antireflexive-relation** (?binary-relation)
relation **irreflexive-relation** (?binary-relation)
relation **reflexive-relation** (?binary-relation)
relation **symmetric-relation** (?binary-relation)
relation **transitive-relation** (?binary-relation)
relation **weak-transitive-relation** (?binary-relation)
relation **equivalence-relation** (?binary-relation)
relation **partial-order-relation** (?binary-relation)
relation **total-order-relation** (?binary-relation)

Object Classification Model (OCM): A Graph-Based Technique for Knowledge Acquisition and Representation

Kunihiko Higa
Information Technology for Management
School of Management
Georgia Institute of Technology
Atlanta, Georgia 30332
Phone 404-894-4365
Fax 404-894-6030
Bitnet: khiga@gtri01

Yihwa Irene Liou
Information and Quantitative Sciences Department
Merrick School of Business
University of Baltimore
Baltimore, Maryland, 21201
Phone 410-625-3420
Fax 410-752-2821
Bitnet: earvlio@ube
Internet: earvlio@ube.ub.umd.edu

ABSTRACT

This paper discusses an Object Classification Model (OCM) as a technique to facilitate the interviewing method to acquire knowledge from a domain expert. In addition, OCM is used to construct knowledge in a graphical format, that can be used to generate production rules. Results of a laboratory experiment comparing knowledge acquisition using interviewing with and without OCM are discussed. This paper concludes with limitations of this technique and future research directions.

1. INTRODUCTION

The expert system development process typically includes four major steps: knowledge acquisition, knowledge representation, implementation, and verification. *Knowledge acquisition*, a vital step, is commonly accomplished through interviews. The lack of effective communication tools to facilitate these interviews causes knowledge acquisition to be very lengthy and time consuming (i.e., usually tape-recording is required and tapes must be transcribed for further analysis and representation). *Knowledge representation* involves transformation of acquired knowledge into a representation scheme, typically production rules. The lack of consistent methods to facilitate the transformation of knowledge causes unreliable and inconsistent knowledge in the knowledge bases (i.e., interpretations of what experts said can be very different). *Implementation* involves building a prototype system utilizing an

existing expert system shell or other tool or language. *Knowledge verification* is a difficult task of removing biases that may be introduced during acquisition and transformation of knowledge.

The purpose of this paper is to introduce an Object Classification Model (OCM) for use in knowledge acquisition, representation, and verification. The rest of the paper is organized into four sections. First, the Object Classification Model is described. Second, an integrated approach to acquire, represent, and verify knowledge using the OCM is discussed. A college admission evaluation system is given to demonstrate OCM. Third, results of a laboratory experiment comparing knowledge acquisition using interviewing with and without OCM are presented. Last, this paper concludes with discussions of limitations and future directions.

2. THE OBJECT CLASSIFICATION MODEL (OCM)

The Object Classification Model (OCM) has been created based on the Structured Object Model (SOM) that was originally developed as an analysis, design, and navigation tool for database applications (Higa and Liu Sheng, 1989) and then adopted for the design of database/knowledge-base integration design (Higa et.al., 1992). The SOM itself is a derivative of the System Entity Structure (SES) that was initially developed as a model representation scheme for simulation modeling (Zeigler, 1987). The OCM represents data semantics using objects, attributes, and two types of relationships: aspect and specialization.

Table 1: Basic Components of OCM

Component	Representation
Object	OBJECT : boldfaced text
Attribute	
Name	*name* : italicized text
Value	value : plain text
Relationship	
Aspect	OBJECT_A (or attribute_x) | OBJECT_B : an object connected as a node.
Specialization	OBJECT_A (or value_x) | *name_b* : an attribute name | typed over a line.
Goal	|| goal value : a goal value connected by a double vertical line.

An object is an event or an activity about which users wish to collect and store information. In OCM, objects are represented by frames. Attributes are used to describe objects by providing them with descriptive properties such as name, ID, etc. Relationships represent associations among objects. Aspects describe owner-property or owner-member relationships. Specializations (i.e., taxonomies) represent the classification/categorization relationships of objects. Table 1 summarizes basic components of OCM and their graphical representations. An example of OCM and associated production rules that represent knowledge of an admission evaluation expert system are given in Figure 1.

The knowledge structuring process of OCM can be decomposed into five general steps which may not be strictly sequential. In Table 2, we discuss these steps using an abstract problem scenario: An organization unit U is responsible for a task T which results in G (goal). An expert system for task T is to be developed.

3. AN INTEGRATED OCM APPROACH

This section presents an approach that uses OCM to integrate knowledge acquisition, representation, and verification in the expert system development process. The method also performs transformation from the graphical forms into production rules. We first review several commonly used knowledge acquisition techniques and compare OCM with them. The semantic networks representation scheme is next described and compared with OCM.

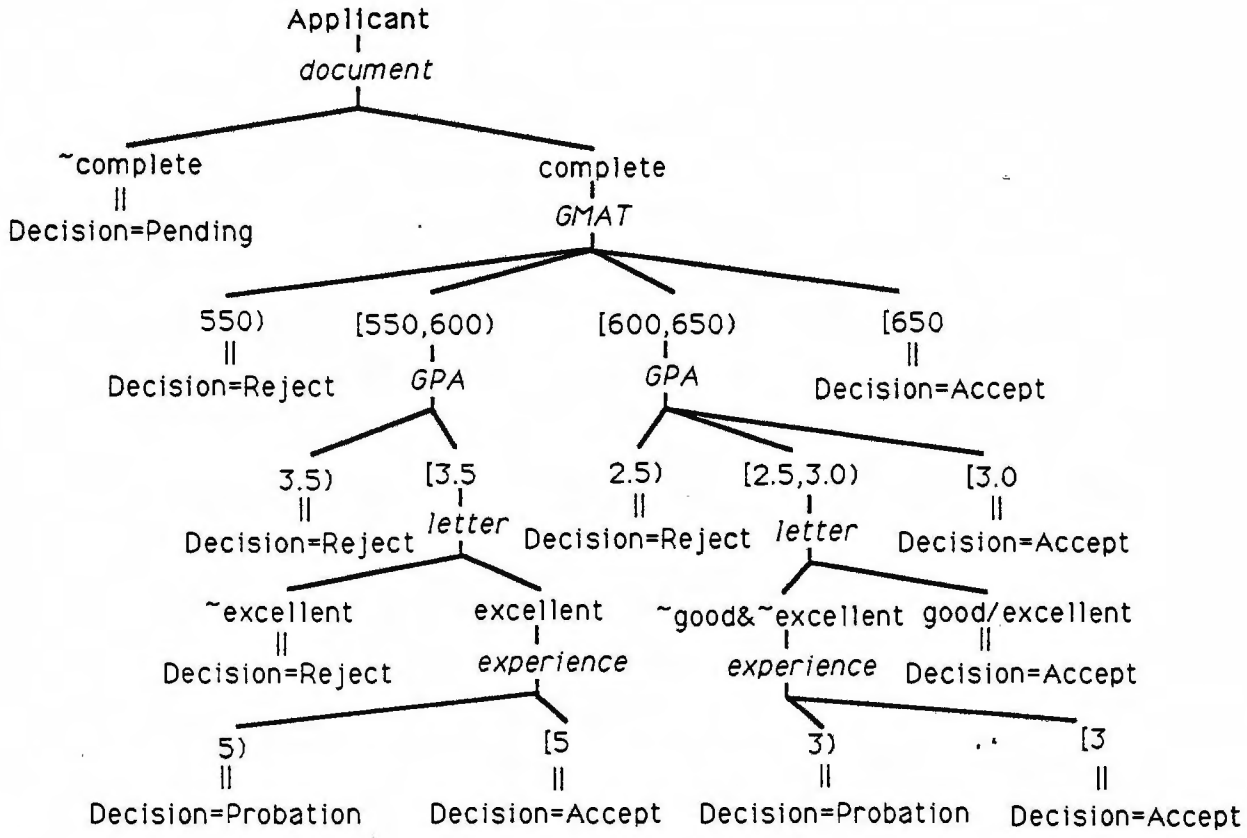
3.1 OCM as a Knowledge Acquisition Method

We briefly review four knowledge acquisition techniques: interviewing, observations, protocol analysis, and repertory grid analysis before examining OCM as a knowledge acquisition method. *Interviewing* is the most common method for eliciting knowledge from domain experts for expert systems development (Olson and Rueter, 1987). The basic form involving question-answer sessions between the knowledge engineer and the expert is very time consuming. Conversations must be captured and be transcribed into transcripts to be analyzed using a supplementary knowledge acquisition method such as protocol analysis (Waterman, and Newell, 1971) or discourse analysis (Abdul-Gader and Kozar, 1990).

An alternative approach, *observations*, allows experts to work in the accustomed environment without interruptions by the knowledge engineer and gives the knowledge engineer insights into the complexities of a problem. However, the underlying reasoning in expert's mind is not revealed in his or her actions. *Protocol analysis* is a form of data analysis that had its origin in psychology. It is usually referred to as "thinking aloud." The expert is asked to describe the thinking process upon solving a problem. This technique can be very time-consuming because protocols must be transcribed and analyzed to determine objects, attributes, relationships, and the kind of inferences drawn from these relationships.

Repertory grid analysis, derived from Kelly's Personal Construct Theory (1955), aims at gaining insights into the expert's mental model of the problem domain. It involves an initial interview with an expert to identify objects in the problem domain and traits that describe these objects, a rating session of the traits, and analyses on the rating results. Although this

Figure 1: OCM for an Admission Evaluation Expert System



Notations:

~: NOT
||: conclusion
(x: x<
[x: x<=
x): <x
x]: <=x
A&B: A and B
A/B: A or B

Decision variables:

document={complete, incomplete};
GPA={0.0..4.0};
GMAT={0..800};
letter={poor, fair, good, excellent};
experience={non-negative integer};

Goals:

Decision={reject, pending, probation, accept}.

Figure 1: OCM for an Admission Evaluation Expert System - continued

Production Rules

Rule 1: IF document = NOT complete THEN Decision = "Pending"
Rule 2: IF document = complete AND GMAT < 550 THEN Decision = "Reject"
Rule 3: IF document = complete AND 550<= GMAT < 600 AND GPA < 3.5 THEN Decision = "Discuss"
Rule 4: IF document = complete AND 550<= GMAT < 600 AND GPA >= 3.5 AND letter NOT = excellent THEN Decision = "Reject"
Rule 5: IF document = complete AND 550<= GMAT < 600 AND GPA >= 3.5 AND letter = excellent AND experience < 5 THEN Decision = "Probation"
Rule 6: IF document = complete AND 550<= GMAT < 600 AND GPA >= 3.5 AND letter = excellent AND experience >= 5 THEN Decision = "Accept"
Rule 7: IF document = complete AND 600<= GMAT < 650 AND GPA < 2.5 THEN Decision = "Reject"
Rule 8: IF document = complete AND 600<= GMAT < 650 AND 2.5 <= GPA < 3.0 AND letter NOT= good AND letter NOT= excellent AND experience < 3 THEN Decision = "Probation"
Rule 9: IF document = complete AND 600<= GMAT < 650 AND 2.5 <= GPA < 3.0 AND letter NOT= good AND letter NOT= excellent AND experience >= 3 THEN Decision = "Accept"
Rule 10: IF document = complete AND 600<= GMAT < 650 AND 2.5 <= GPA < 3.0 AND (letter = good OR letter = excellent) THEN Decision = "Accept"
Rule 11: IF document = complete AND 600<= GMAT < 650 AND GPA >= 3.0 THEN Decision = "Accept"
Rule 12: IF document = complete AND GMAT >= 650 THEN Decision = "Accept"

Table 2: Steps of Knowledge Structuring Process

Step	Description	Results
1	Identify the task related entities, goals, their domains, and people who perform the task.	P (local expert) performs T using a, b, c. Domains are: a={a1, a2, a3}; b=[Lo_b..Hi_b]; c={c1, c2}; G={g1, g2, g3}
2	Construct an initial knowledge structure for the task.	For g1, a must be either a1 or a2 and b must be less than b1 and c must be c1. For g2, a must be a2 and b must be greater than b2 and less than b3 and c must be c2. For g3, a can be any value from the domain and b must be greater than b4 and c can be either c1 or c2. A SOM diagram is shown in Figure 2.
3	Ask local experts (people identified in Step 1) to examine the knowledge structure and modify it if necessary. If modification is necessary, record reasons for each modification.	Suppose P modified <a2, (b2, b3), c2> -> g2 to <a2, [b1, b4], c2> -> g2
4	Check the knowledge structure for completeness and consistency. Make necessary modifications to the knowledge structure and then repeat Steps 3 and 4 until no further modification is required.	At this point, the knowledge structure is incomplete and possibly inconsistent because: (1) c2 is not considered at < b1 branch; (2) less than and equal to b4 is not considered at a3 branch; (3) if b1 < b4 then [b1, b4] is not considered at a1 and a3 branches; (4) if b1=b4 then "equal to b1 (or b4)" is not considered at a1 and a3 branches; (5) if b1 > b4 then inconsistency exists at a1 and a2 branches. These observations are discussed with P and steps 3 and 4 are repeated.
5	Translate the knowledge structure into rules (and modify the organization model if necessary).	After all modifications are completed, the knowledge structure is translated to rules. Figure 3 shows the final knowledge structure and corresponding rules.

Figure 2: Initial Knowledge Structure Diagram

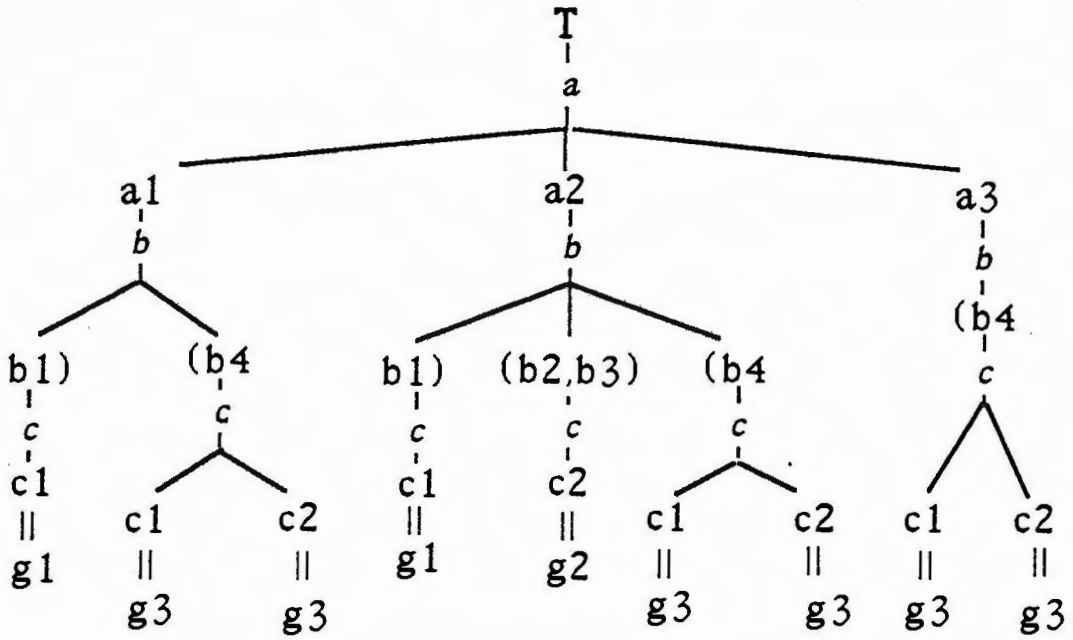
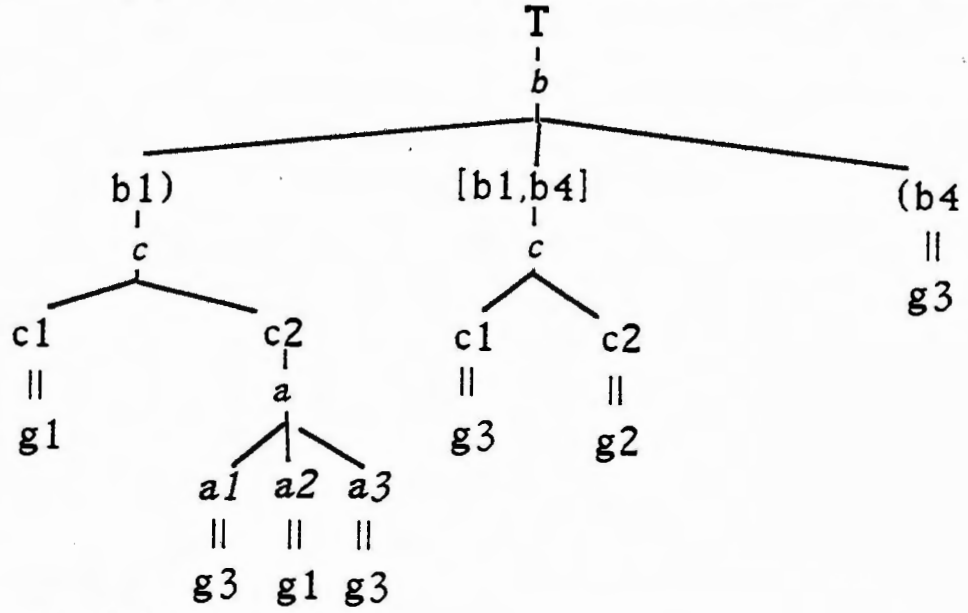


Figure 3: Final Knowledge Structure and Corresponding Rules



Production Rules

Rule 1: IF b < b1 AND c = c1 THEN g1
Rule 2: IF b < b1 AND c = c2 AND a = NOT a2 THEN g3
Rule 3: IF b < b1 AND c = c2 AND a = a2 THEN g1
Rule 4: IF b >= b1 AND b =< b4 AND c=c1 THEN g3
Rule 5: IF b >= b1 AND b =< b4 AND c=c2 THEN g2
Rule 6: IF b > b4 THEN g3

technique is useful in extracting subjective data, it can be very time-consuming.

OCM can be used with a structured interviewing technique where question-answer sessions are used to elicit objects, attributes, and relationships about the problem domain. For instance, an expert of college admission evaluation might be asked "What are the requirements?" "What is the minimum GPA required?" "What is the minimum GMAT score required?". An OCM diagram is sketched and is presented to the expert during the interview session. This allows the expert to review knowledge structures and provide immediate feedback without transcription and analysis of verbal protocol. Furthermore, the OCM diagrams demonstrate their decision rationale and serve as a communication tool between the knowledge engineer and the expert(s). See Figure 1 for an OCM representing admission evaluation knowledge.

3.2 OCM as a Knowledge Representation Scheme

OCM is similar to Semantic Nets (Harmon and King, 1985) in that both identify objects/entities and attributes of those objects. Semantic Nets, also called Semantic Networks, is a knowledge representation scheme based on a network structure that has nodes connected by links representing the relations between the nodes. Nodes can be objects or attributes. Three types of links are available in semantic nets: "is_a" links represent instance relationship; "a_kind_of" links represent class relationships; and "has_a" links represent owner-property relationships. Similarly OCM also allows property and classification relationships to be specified in the diagram; however, unlike semantic nets, a set of rules that represent the target knowledge can be mechanically generated from OCM.

The proposed OCM is also very similar to a decision tree. However, a decision tree in its current form cannot be used for designing and maintaining rule bases in the same manner as OCM because there is no distinction between aspects and specializations of objects in a decision tree. This makes checking of incompleteness of rule bases difficult. For example, when a specialization is used in OCM, all subclass objects (i.e., specialized objects) must be represented; however, this restriction does not apply when an aspect is used in OCM (all aspect objects may or may not be used). Therefore, a decision tree is inappropriate for the design and analysis of rule bases that contain multiple related objects with multiple specializations.

In summary, OCM as a knowledge representation scheme provides the following benefits: (1) It provides an integration with the knowledge acquisition process if OCM was used to acquire knowledge from the expert. (2) A hierarchical knowledge structure may be used to demonstrate the complete knowledge base. (3) Knowledge structures can be directly transformed into rules. (4) Consistency and completeness of the knowledge base may be examined by OCM.

3.3 OCM for Verification and Integration

In addition to representing knowledge in graphical form for direct transformation into production rules, OCM is also used to verify acquired knowledge. Two types of verification can be performed in OCM. A simplified verification involves showing the OCM diagrams to the expert and ask him/her to identify incorrect representations. A more complexed

verification involves the consistency and completeness checking of the knowledge base (i.e., rules).

OCM offers the following benefits for the expert systems development: (1) graphical knowledge structures as a communication vehicle between a knowledge engineer and a local expert; (2) direct transformation from knowledge structures into rules; (3) integrates the knowledge acquisition and the knowledge representation phases; (4) improves the reliability of the knowledge base; and (5) a vehicle for systematic verification for the acquired knowledge.

4. A TEST OF OCM

To evaluate the usefulness of OCM for knowledge acquisition, we conducted a laboratory experiment involving students in expert systems classes from Georgia Institute of Technology and University of Baltimore working on a project relating to evaluating applications to MBA programs. Figures 1 and 4 jointly represent the solution of this application. In Figure 1, rules for the admission decision are depicted. Figure 4 represents rules for the financial aid recommendation. The experiment was conducted during Fall 1991 and Spring 1992.

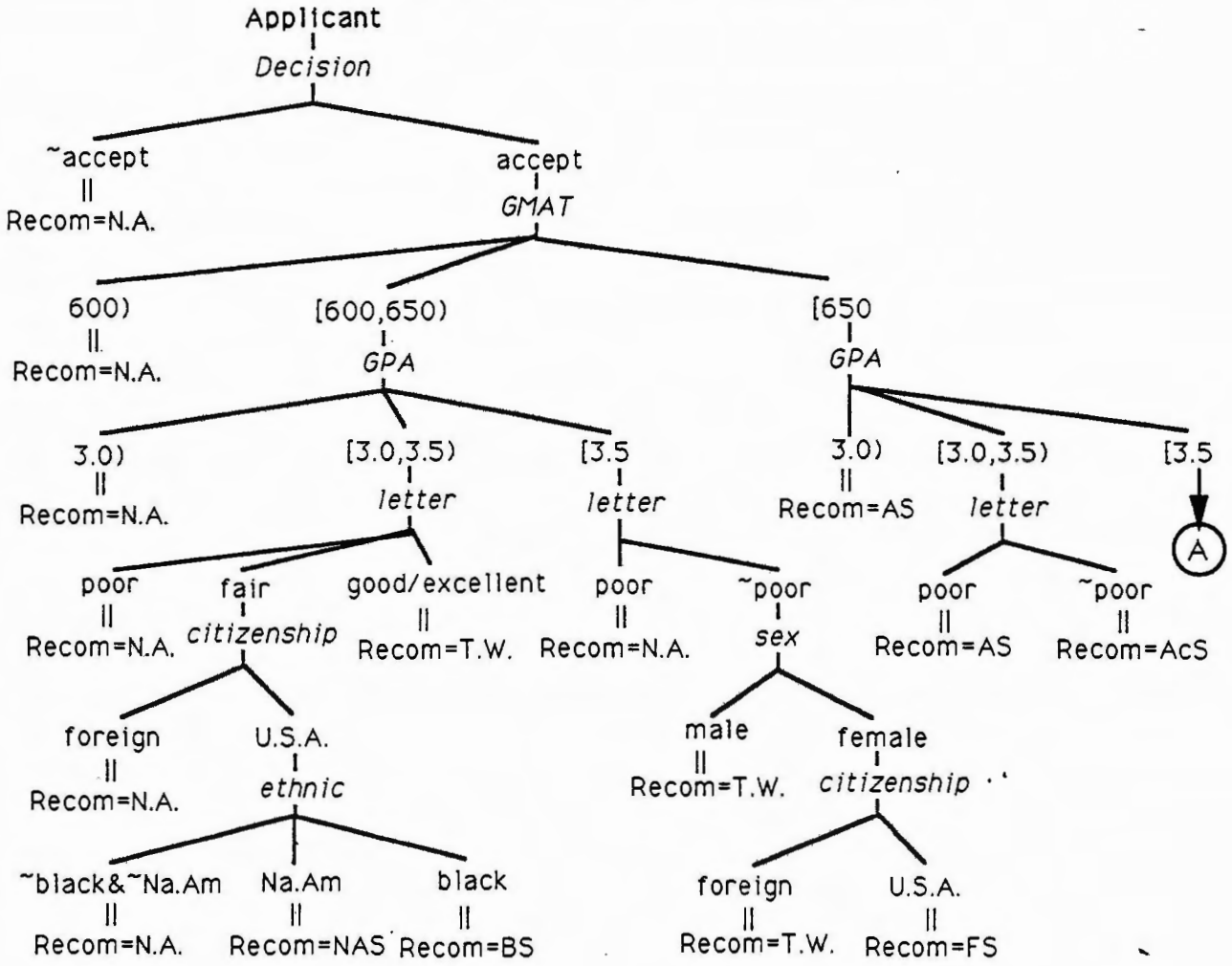
4.1 Pilot Study

Twenty student from an upper level undergraduate expert systems class at Georgia Institute of Technology in Fall 1991 participated in the pilot study. Four students who had no work experience were trained to be domain experts. The remaining 16 students were grouped into two groups, one half used traditional interviewing technique while the other half used the graph-based technique (i.e., OCM). Interviewing technique, concept sorting, and production rules were taught to all 20 students, and the graph-based technique was taught to those assigned to that group. Four students (two from each group) were assigned to each domain expert. The task was to acquire knowledge from the assigned domain expert individually and represent the domain expert's knowledge using production rules.

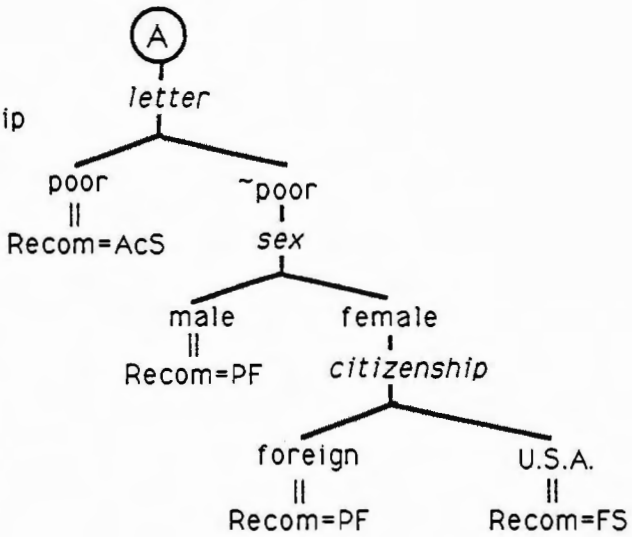
Each student had 7 days to complete the task and there was no limitation to the number of meetings s/he could have with the assigned domain expert and how much time they could spend at each meeting. A post-project questionnaire was administrated to collect students' perceived satisfaction with the knowledge acquisition process. Perceived satisfaction with the knowledge acquisition process was measured by the following items: (1) student's experience in completing the task (enjoyable or frustrated); (2) satisfaction with student's performance of the task; (3) efficiency of the process; (4) time taken for obtaining solution for the task; and (5) feedback regarding the graph-based technique.

In addition, the acquired knowledge and production rules were evaluated based on the quality of rules. The quality was measured by the following items: (1) number of rules missed (If a certain branch of the OCM tree is not represented, then it was counted as missed); (2) number of contradiction among rules, e.g., R1: IF A THEN B, R2: IF A THEN Not B, either rule may or may not be correct, but they contradict to each other; (3) number of inconsistent rules (with facts), e.g., R1: IF A THEN B, the truth is "Not B"; (4) number of duplicated rules (i.e., identical rules); (5) number of overlapped rules, e.g., R1: IF $x < 10$ THEN Y, R2: IF $x < 5$ THEN Y; (6) number of complemented rules, e.g., R1: IF A and C THEN B, R2: IF

Figure 4 OCM for Financial Aid Recommendation



Notations: N.A.: Not Available
T.W.: Tuition Waiver
FS: Female Student Scholarship
NAS: Native American Student Scholarship
BS: Black Student Scholarship
AS: Assistantship
AcS: Academic Scholarship
PF: Presidential Fellowship
Na.Am: Native American
Recom: Recommendation



Decision variables:
Decision={reject, pending, probation, accept};
GPA={0.0..4.0};
GMAT={0..770};
letter={poor, fair, good, excellent};
citizenship={foreign, U.S.A.};
sex={male, female};
ethnic={Native American, Black, Caucasian, Asian, Hispanic, Others}

Goals: Recom={N.A., NAS, BS, T.W., AS, AcS, FS, PF}.

Not A and C THEN B, in this case, R1 and R2 can be combined to be "IF C THEN B."

Table 3 depicts the total number of rules identified in each category. It is apparent that the graph-based group results in much less number of rules missed and contradicted than that of traditional group. Table 4 presents descriptive statistics of perceived satisfaction with the knowledge acquisition process on a scale of 1 to 7.

4.2 Second Study

Twenty-nine graduate students enrolled in the expert systems class at University of Baltimore in Spring 1992 participated in the study. All 29 students were taught interviewing technique, concept sorting, protocol analysis, observations, and questionnaires. In addition, knowledge representation schemes including decision trees, decision tables, production rules, semantic nets, and object-attribute-values were discussed. Four students were randomly assigned and trained to be domain experts. The remaining 25 students were assigned randomly into two groups: 12 for traditional group and 13 for graph-based group. Three students from each group were then assigned to work with one domain expert (one domain expert worked with 7 students: 3 of traditional group and 4 of graph-based group). The task was the same as the pilot study. Post-project questionnaires were administrated and production rules submitted by each individual were evaluated according to the same set of criteria. Tables 5 and 6 present results.

4.3 Discussions

Results of both studies indicate that groups applying the graph-based technique performed better than groups employing the traditional group in terms of quality of the knowledge base. The graph-based technique especially contributes to the consistency and completeness of the knowledge base. In addition, the graph-based technique is more efficient and easy to work with and the process is more enjoyable.

One drawback of OCM which was noted by several subjects after the experiment is the cumbersomeness of modification on the OCM diagram. Because OCM diagrams were drawn by hands in this experiment, modifications often resulted complete or major redrawing of the diagram. Many subjects commented on the definite needs for software support (i.e., a CASE tool) for OCM.

5. LIMITATIONS AND FUTURE DIRECTIONS

There are some limitations of the integrated OCM method. First, the method provides transformation only to the rule-based expert systems. Second, although, the proposed method may be effective for design of large scale rule bases, it works only for classification knowledge applications at this point. In a non-classification application, multiple instances of a object class are typically competing for limited resource. For example, a classroom assignment application requires all courses to be assigned to some classrooms. However, each course will have certain requirements (e.g., time, capacity, and equipment) and each room will have certain constraints (e.g., availability, capacity, and facility). Therefore, this application requires not only classification of each course but also conflict resolution between competing courses. A way to handle non-classification applications must be studied for the OCM

Table 3: Pilot Study - Total Number of Rules with Problems

	Traditional	Graph-based
Number of rules missed	16	6
Number of rules contradicted	14	5
Number of inconsistent rules	5	2
Number of duplicated rules	4	0
Number of overlapped rules	0	0
Number of complemented rules	6	3

Table 4: Pilot Study - Perceived Satisfaction with KA Process

	Traditional	Graph-based
Experience in completing the task (is enjoyable)	3.87	4.37
Satisfaction with your performance of the task	5.12	5.5
Efficient process for obtaining solution for the task	3.12	5.12
Reasonable time taken for obtaining solution for the task	3.37	4.87
Graph-based technique is easy to use	N/A	4.5
Use graph-based technique to complete a similar task	N/A	4.87

*Rating on a scale of 1-7.

Table 5: Second Study - Total Number of Rules with Problems

	Traditional	Graph-based
Number of rules missed	23	3
Number of contradiction of rules	15	5
Number of inconsistent rules	8	0
Number of duplicated rules	7	0
Number of overlapped rules	0	0
Number of complemented rules	10	0

Table 6: Second Study - Perceived Satisfaction with KA Process

	Traditional	Graph-based
Experience in completing the task (is enjoyable)	4.60	5.40
Satisfaction with your performance of the task	5.90	5.73
Efficient process for obtaining solution for the task	5.80	5.90
Reasonable time taken for obtaining solution for the task	4.60	5.53
Graph-based technique is easy to use	N/A	5.87
Use graph-based technique to complete a similar task	N/A	5.73
Understood what was needed when completing the task	5.30	5.53
Not frustrated when completing the task	5.10	5.47

* Rating on a scale of 1-7.

method.

We have proposed an approach to integrate the knowledge acquisition, representation, and verification of expert systems development using the OCM method. Components of the OCM were discussed. Examples were given to demonstrate the applicability of such a method. The knowledge structuring process is described. A laboratory experiment was conducted to verify the effectiveness and usefulness of the method. The transformation of OCM into frames and other representation schemes deserves additional attention. To further expand this method to systems development in general, we realize that a computer-aided knowledge engineering tool would be useful in the future to support many of the analysis, design, and implementation activities.

REFERENCES

Abdul-Gader, A.H., Kozar, K. (1990). Discourse analysis for knowledge acquisition, *Journal of Management Information Systems* 6:4:61-82.

Harmon, P., and King, D. (1985). *Expert Systems for Business*, Wiley, New York.

Higa, K, and Liu Sheng, O. R. (1989). An object-oriented methodology for an end-user logical database design: the Structured Entity Model approach. *Proceedings of IEEE 13th International Computer Software & Applications Conference*, Orlando, FL, September 19, 365-373.

Higa, K., Morrison, C.M., Morrison, J., and Liu Sheng, O.R., An Object -Oriented Methodology for Knowledge Base/Database Coupling, *Communications of the ACM*, 35:6, June 1992.

Kelly, G. (1955). *The psychology of personal constructs*, Norton, New York.

Olson, J., and Rueter, H. (1987). Extracting expertise from experts: methods for knowledge acquisition, *Expert Systems* 4:3:152-168.

Waterman, D.A., Newell, A. (1971). Protocol analysis as a task for Artificial Intelligence, *Artificial Intelligence,* 2:285-318.

Zeigler, B. (1987). Knowledge representation from Minsky to Newton and beyond, *Applied Artificial Intelligence,* 1:1:87-107.

The Active Glossary: Taking Integration Seriously

Georg Klinker, David Marques, John McDermott, Therese Mersereau, and Linda Stinson

Digital Equipment Corporation
111 Locke Drive
Marlboro, MA 01752

klinker@airg.dec.com

Developing support for any task involves analyzing a workplace, designing a problem-solving approach and knowledge base, populating that knowledge base with information required by the problem-solving approach, and introducing the new support into the workplace. Knowledge-acquisition tools developed to date support only a subset of these activities. They are not integrated into a coherent knowledge-engineering environment. A current trend in knowledge acquisition is to move towards knowledge-engineering environments supporting the entire solution-development life cycle. This paper introduces the Active Glossary, a tool that is part of the Spark, Burn, FireFighter knowledge-engineering environment. The Spark, Burn, FireFighter tools assist a knowledge engineer in mapping parts of a workplace description to other parts of the same description and to pre-defined programming constructs. The Active Glossary has been integrated with a tool that assists analysts with describing a workplace and programmers with describing programming constructs.

1. Introduction

The process of adding support -- automated or not automated -- for a task is a complex one involving many overlapping activities that have to be carefully managed and integrated. The activities include analyzing the workplace, designing a problem-solving approach and knowledge base, populating that knowledge base with information required by the problem-solving approach, and integrating the new support into the workplace. Knowledge-acquisition tools developed to date support only a subset of these activities, and rarely share the information gathered during each of the activities.

1.1. Some Approaches to Knowledge Acquisition

The following concisely characterizes existing knowledge-acquisition tools and approaches with respect to the knowledge-engineering activities they support. In this paper, we use the term "knowledge engineering" to refer to the entire process of adding support for a task. The support can be automated, that is, it can be a program, or it can be not automated. The knowledge-engineering process is performed by a "knowledge engineer": a generic term denoting a person or, more likely, a group of people helping the workers in the workplace to introduce support. We use that term when we do not want to differentiate between the different roles knowledge engineers play in the solution-development life cycle.

The intention of this section is not to provide a complete taxonomy. See [Boose 1989] and [Westphal et al. 1989] for a comprehensive compilation of existing knowledge-acquisition tools. Also, we do not include software engineering approaches in this summary. See section 4 for some references to work in software engineering. The following characterization outlines current knowledge-engineering approaches with the goal of showing when these approaches can be useful in the life cycle of expert systems.

1.1.1. Analyzing the Workplace

The roles that a knowledge engineer must assume include that of an analyst. An analyst performs the "task analysis". Although the term task analysis is often used by researchers in knowledge acquisition to refer just to the determination of whether the problem is "classification" or "abduction" or "scheduling", it more properly refers to the process of creating, making explicit, and structuring a description of the work done in a workplace to accomplish a particular task. That is the sense in which we use it here. Analysts are knowledge engineers who work with the people in the workplace to create a workplace description. In this paper, we use the term analyst to refer to a knowledge engineer with analysis skills. An analyst does not need to be a programmer. Furthermore, a good analyst performs a task analysis without considering how to create a program implementing the task. That is difficult to do for a programmer.

When presented with a new task, a group of knowledge engineers must perform a workplace analysis to determine a reasonable working definition of the activities in the workplace. During this analysis the knowledge engineers need to uncover detailed information about relevant activities, including the domain's essential concepts and vocabulary, the activities performed in the workplace, the resources consumed and the products produced, the agents performing the activities, and many more. Conceptual frameworks for performing such analysis include Newell's knowledge level [Newell 1982]. The objective of a task analysis is to determine the essential knowledge required for a task, without considering the way in which the task will be performed.

For example Balkany and colleagues [1991] identify the functions needed for design (constructive) tasks by performing successive decomposition. The decomposition proceeds until a set of operators are found that operate on a predetermined, albeit general, knowledge structure. By predefining the knowledge structure, the grain size of the operators is uniform. The authors indicate that performing knowledge-level analysis can reduce system development by identifying knowledge that can be reused from existing systems and combined in different ways.

Several tools assist a user in uncovering initial concepts, as well as in identifying structural and functional relationships for the domain. These tools can be used as part of a comprehensive task analysis. For example, ETS [Boose 1984] and KSSO [Shaw et al. 1987] are tools based on personal-construct theory. They assist users in determining structural relationships and relevant terms and concepts using the repertory-grid technique. They place terms on a grid and interact with a user to identify clusters. As clusters are found, concepts and relationships among them are determined. KRITON [Diederich et al. 1987] combines a set of knowledge-elicitation techniques, such as text analysis, to uncover initial concepts and structural relationships; interview techniques to refine the results of the text analysis; and protocol analysis to determine functional relationships. SOK [Linster 1989] extends KRITON to integrate additional elicitation techniques with a modeling component that describes the problem-solving process.

1.1.2. Designing a Problem-Solving Approach and Knowledge Base

Based on the task analysis, knowledge engineers design an approach to perform the task and a representation scheme for the knowledge required by the problem solver. Programmer is another role that a knowledge engineer must assume when adding automation into the workplace. A programmer translates a workplace description into a collection of mechanisms automating parts of the activities in the workplace. In this paper we use the term programmer to

refer to a knowledge engineer with programming skills. A programmer does not need to be an analyst.

Researchers in knowledge acquisition developing tools to support with this design task are mainly concerned with developing conceptual frameworks to guide the implementation process. For example, Steels introduces a comprehensive framework for translating a knowledge-level description to code [Steels 1990]. That is, his framework describes how to design a knowledge-based system from a knowledge-level analysis. Design activities are performed at a level in between the knowledge level and the code level; Steels calls this level the knowledge-use level. His framework integrates different aspects of the design task: decomposing a task into subtasks, categorizing problem-solving approaches and the kinds of knowledge they require, and organizing domain-specific knowledge.

Other researchers have developed frameworks that focus on specific aspects of the design task. For example, the generic tasks approach [Chandrasekaran 1990] describes how to decompose an application into general task categories. McDermott emphasizes the problem-solving approach of an application and describes a variety of problem-solving methods and knowledge roles required to support each method [McDermott 1988]. Balkany and colleagues [1991] take a similar approach; they have reported an initial set of building blocks for problem-solving methods that are relevant for configuration design. They derived this set by analyzing the operation of existing design tools.

Another group of researchers has made progress characterizing individual, problem-specific problem-solving approaches and the tasks that such approaches can solve. Examples are approaches to solve reporting tasks [Klinker 1989], skeletal plan refinement tasks [Friedland 1989], classification tasks [Clancey 1985], and configuration design tasks [Runkel et al. 1992].

1.1.3. Populating a Knowledge Base

Once the problem-solving approach and the structure of the knowledge base are defined, the knowledge base needs to be populated. A variety of tools have been developed assisting a knowledge engineer with this task. These tools derive their power by presupposing the problem-solving approach and the organization of the associated knowledge base. That is, they can only be used once the problem-solving approach and the structure of the knowledge base are known. Furthermore, a given approach and structure must match the one presupposed by the tool under consideration.

Examples of knowledge-acquisition tools with explicit problem-solving models are: MOLE [Eshelman et al. 1987] for selective tasks; BURN [Marques et al. 1988] for sizing tasks; SALT [Marcus 1988] and CGEN [Birmingham 1988] for constructive tasks; KNACK [Klinker 1988] for reporting tasks; and OPAL [Musen 1989] for the task of entering cancer-therapy protocols. All of the tools that are listed above have the explicit goal of interacting directly with a domain expert, that is, interacting with somebody who is not necessarily a programmer.

1.1.4. Introducing New Support into the Workplace

Adding new support to a workplace introduces change. If the new support is introduced without consideration of the way it will interact with other agents and information sources in the workplace, it will not fit well, and will probably not be used. Research in knowledge acquisition and software engineering has typically ignored this problem [Zuboff, 1988].

The introduction of new support into a workplace requires knowledge about the workplace, and must be integrated with workplace-analysis activities. Considering the details of how new support will be introduced into the workplace affects the information gathered during workplace analysis. In particular, this forces attention to the conditions under which each activity is started and the range of normal exceptions that must be handled when resources or agents are not available.

Most of these dependencies do not emerge until the new support is used in the workplace. This can result in further development of the workplace description and introduction of additional support.

1.2 Towards Knowledge-Acquisition Environments

The approaches to knowledge acquisition described above are not concerned with integrating individual tools across the life cycle of an expert system. It is up to the knowledge engineers to determine when to use which tool and transfer the results of an interaction from one tool to another. None of the knowledge-acquisition tools that have been developed to date can support a group of knowledge engineers throughout the life cycle of a knowledge-based system. Different tools support different aspects of a knowledge engineer's task.

A current trend in knowledge-acquisition is aimed at providing a support environment for integrating all of a knowledge engineer's activities. One example is the Spark, Burn, FireFighter project [Marques et al. 1992; Dallemagne et al. 1992]. This project is defining mechanisms: building blocks for problem-solving methods. The approach is to build mechanisms that are specific enough to allow the use of powerful knowledge-acquisition tools for instantiating them, yet small enough to allow recombination, increasing their applicability. The Spark, Burn, FireFighter project presupposes a library of such mechanisms and associated knowledge-acquisition tools as well as a framework for configuring and exploiting them. The framework consists of tools that interpret between the knowledge engineer (who is not necessarily a programmer) and the mechanisms. It includes tools to represent workplace descriptions, assist with the management of the activities in the workplace, identify mechanisms that might provide assistance in that workplace, customize and introduce those mechanisms into the workplace, and help respond to change in the workplace.

For this knowledge engineering environment, integration is a major issue. It is not helpful for the knowledge engineers to just provide them with a set of tools and leave it up to them to use the right tool at the right time with the right data. Using each tool in isolation only increases the demands put on the knowledge engineers when developing a knowledge-based system. Integrating the tools includes providing assistance to the knowledge engineers in using the tools together. Integration is achieved when it is easier to use the tools together than to use each in isolation.

This paper introduces the Active Glossary, a tool that assists in mapping the results of a task analysis to mechanisms. The Active Glossary has been integrated with the Spark tool and a CASE tool; the CASE tool helps with documentation efforts and Spark assists a knowledge engineer with representing a workplace description and mechanism description. Section 2 describes the Active Glossary and briefly describes Spark in order to point out some integration issues. Section 3 provides some data on the use of the Active Glossary. A brief description of related efforts in knowledge-acquisition and software-engineering research is presented in Section 4. Section 5 outlines some integration requirements for knowledge-acquisition tools that take integration seriously.

2. The Active Glossary: An Example

Given a description of a workplace, the Active Glossary assists a knowledge engineer with making different descriptions of the workplace consistent and with identifying mechanisms that could automate a given activity in the workplace. To accomplish this, it facilitates the use of a shared vocabulary or glossary that describes a workplace using an analyst's vocabulary, and that describes mechanisms using a programmer's vocabulary.

Part of an analyst's job is to uncover structure in the workplace. This includes determining to what extend any two activities are the same. Real life tasks are very complex. They involve different activities, different relationships between activities, different interrelationships with other tasks, and different people. These differences are so important to the people involved that no two tasks will seem quite the same to them. The Active Glossary assists an analyst with making different perspectives of the workplace consistent. Its approach is to facilitate the creation of a vocabulary for describing task activities that is common to multiple activities across tasks. Workplace activities can then be related to that shared vocabulary. Two different activities are considered similar if they are related to the same set of terms in the glossary.

To improve a workplace based on an organization's business goal, an analyst might recommend automating some of the workplace activities. This includes determining whether an existing program can be reused to automate a given activity in the workplace. The Active Glossary assists a knowledge engineer and the workers with making that decision: if two workplace activities are considered the same, the programming constructs implementing them are the same and can be reused for the activity at hand. In other words, if an activity's shared vocabulary has been previously associated with programming constructs, the programming constructs can be used to implement the activity.

A very important problem with the use of a shared vocabulary, taxonomy, or ontology is that we cannot assume that people will understand somebody else's vocabulary without any effort. That vocabulary must be learned. The point is that when we advocate the use of a shared vocabulary we must also provide help to other people learning that vocabulary. The Active Glossary helps its users learn the shared vocabulary. If an analyst or programmer cannot identify the shared vocabulary for a workplace activity to be automated or a program to be described, the Active Glossary facilitates the process by providing different ways of learning about the existing terms. For example, it suggests relevant terms, it suggests the relationship between a term and other terms within its class, and it illustrates a term's use for describing other workplace activities.

The Active Glossary presupposes a description of a workplace and a description of mechanisms. A knowledge engineer creates those descriptions using the Spark tool. Therefore, the Active Glossary is integrated with the Spark tool.

The following example illustrates the Active Glossary. It demonstrates the advantage of exploiting some context information from a workplace analysis to assist with structuring other information in the workplace description; to assist with designing the problem-solving approach; and to assist with populating the knowledge base. Finally, the context information is used to introduce the newly created support into the workplace, such, that it works with other automated or non-automated activities already in the workplace.

2.1. Analyzing the Workplace: Assistance in Structuring the Information

In this example, an analyst is using the Spark tool to describe a workplace. Spark assists an analyst in representing the information about a workplace that is necessary to (1) select programming constructs automating some of the workplace activities and (2) provide information to a workflow controller managing the activities in the workplace.

Figure 1 shows an overview presenting the current state of the workplace description. A graphical representation of a part of the workplace is called a "view". The view in Figure 1 is generated automatically from the current representation of the workplace. It summarizes the workplace description with the goal of helping the user navigate through the description. This navigation view shows how the workplace has been subdivided into several views. The division describes the workplace from the perspective of four people. Each box stands for an activity view (see Figure 2). The boxes to the right detail the boxes to the left.

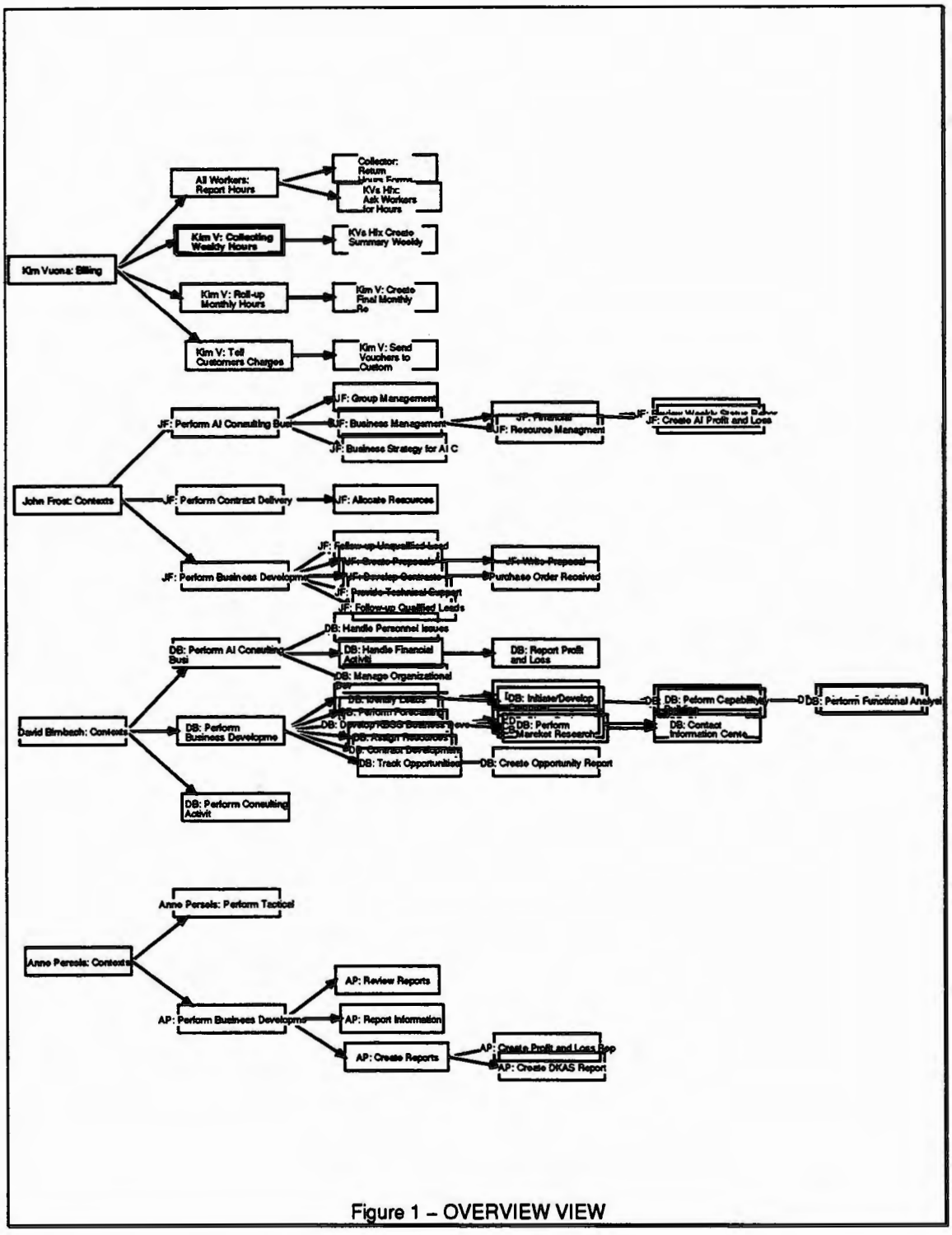


Figure 1 – OVERVIEW VIEW

Continuing the example, the analyst wants to extend the description for the billing process (Kim Vuona: Billing). Kim Vuona is responsible for collecting the weekly hours from each employee, rolling them up into a weekly report, creating a monthly report, and determining the charges for the customers on a monthly basis. The analyst wants to refine the description of creating the weekly report and clicks on the appropriate box from the overview picture (Kim V: Collecting

Weekly Hours). This results in an activity view of the workplace detailing some of the billing task as shown in Figure 2.

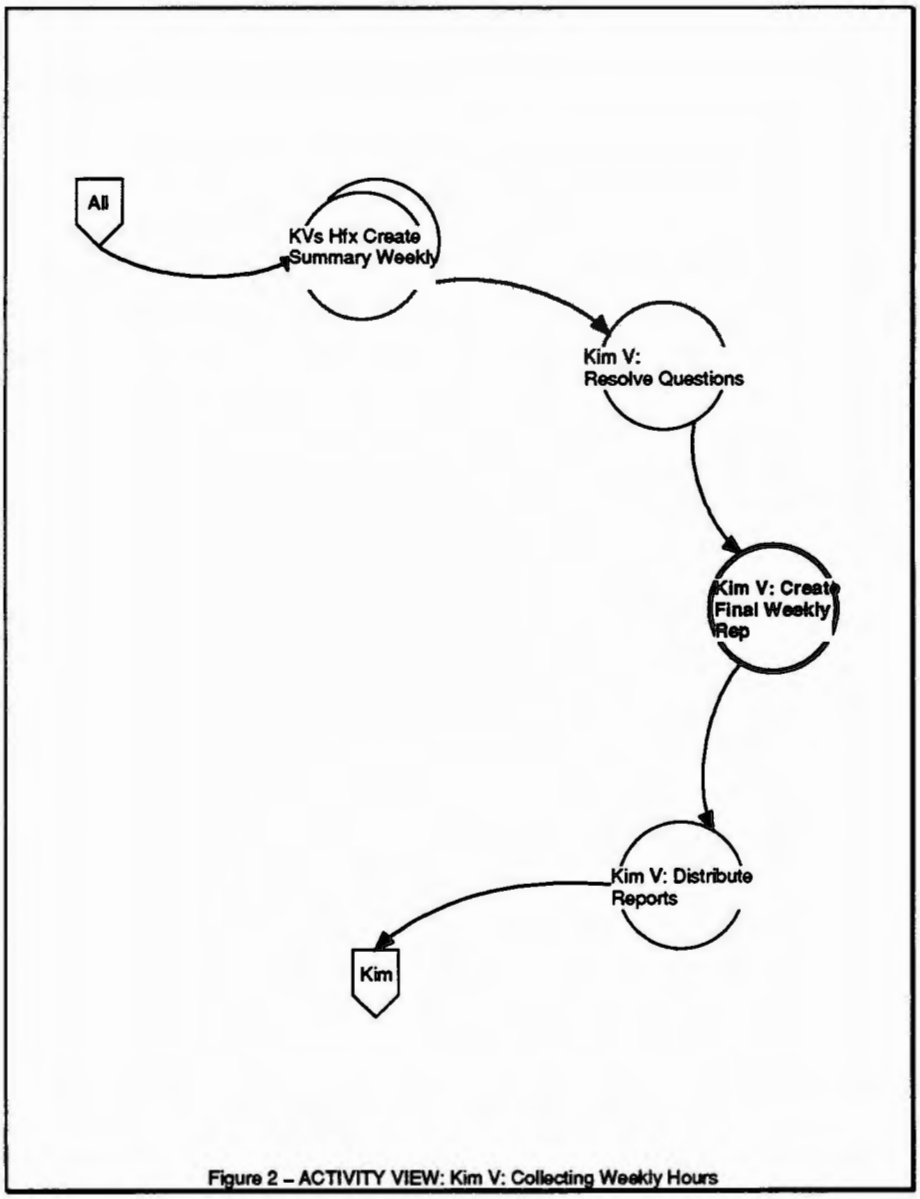


Figure 2 – ACTIVITY VIEW: Kim V: Collecting Weekly Hours

"Collecting Weekly Hours" consists of creating an initial report, refining that report, creating the final report, and distributing that report. Figure 3 summarizes the information that was defined for one of the activities shown in Figure 2: "Kim V: Create Final Weekly Report".

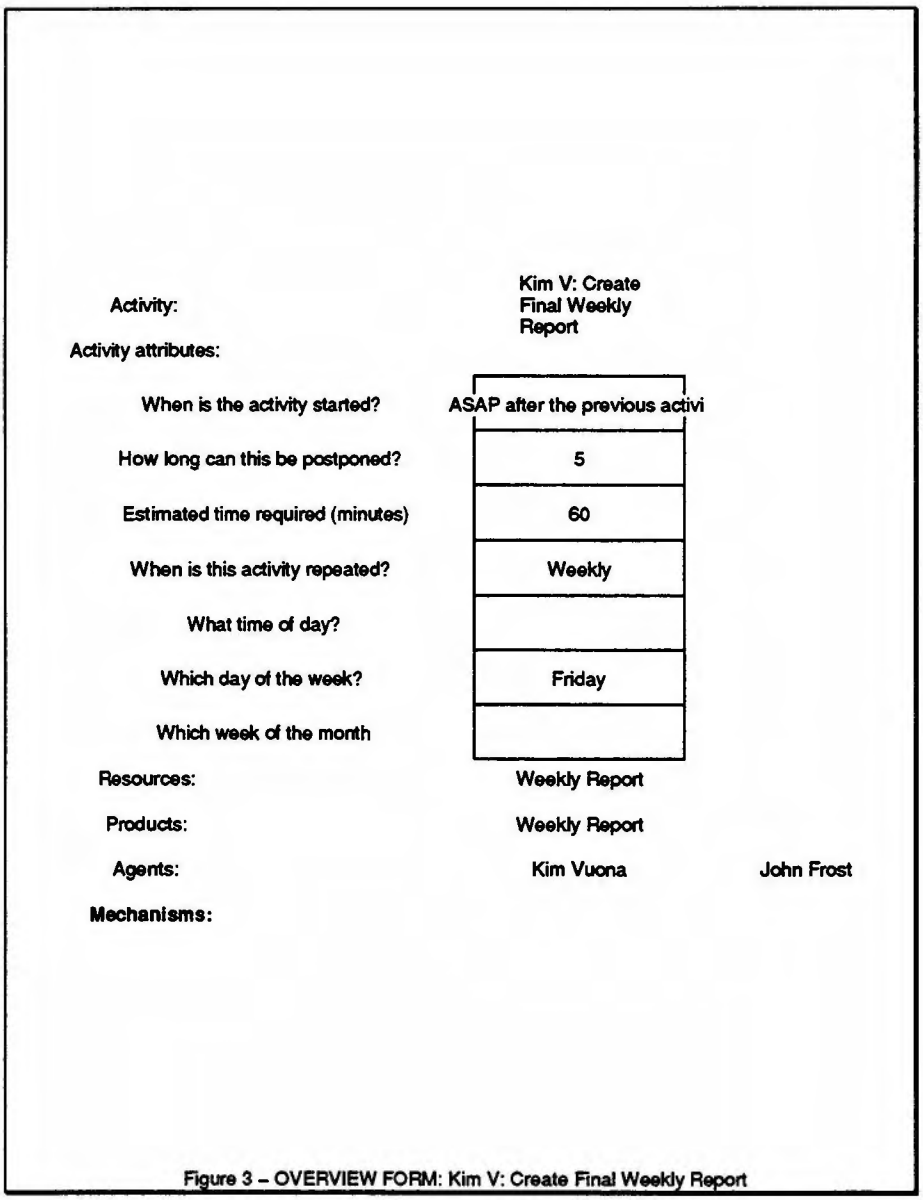


Figure 3 – OVERVIEW FORM: Kim V: Create Final Weekly Report

Figure 3 represents another view that is generated automatically from the current description of the workplace. Here, the goal is to summarize all the information that is relevant for the "Kim V: Create Final Weekly Report" activity. This includes attributes for the activity, agent, all resources used, and for the products. For example, the activity attributes shown in Figure 3 define when to start an activity, how long it will take, how often it is performed, etc. This information is used by a workflow controller managing the activities in the workplace. The workflow controller determines when an activity can be performed by checking whether all resources are available that are required by an activity, and by ensuring that time constraints are satisfied. If an activity can be performed, the workflow controller sends messages to the agents performing that activity. The messages start a program, if the activity is performed by an automated agent, or send electronic mail to the agents involved.

In our example, given this current description of the workplace, the analyst wants assistance in automating the "Kim V: Create Final Weekly Report" activity from Figure 2. This request is passed on to the Active Glossary.

2.2. Designing a Problem-Solving Approach

In order to select programming constructs that might automate the "Kim V: Create Final Weekly Report" activity, the Active Glossary needs some information from the Spark tool.

2.2.1. Exploiting Status and Context Information

The status of that information as well as the steps an analyst must follow to define that information, is presented in an interface view shown in Figure 4. It presents the interface between the Spark tool and the Active Glossary and allows the analyst to easily move back and forth between the two tools.

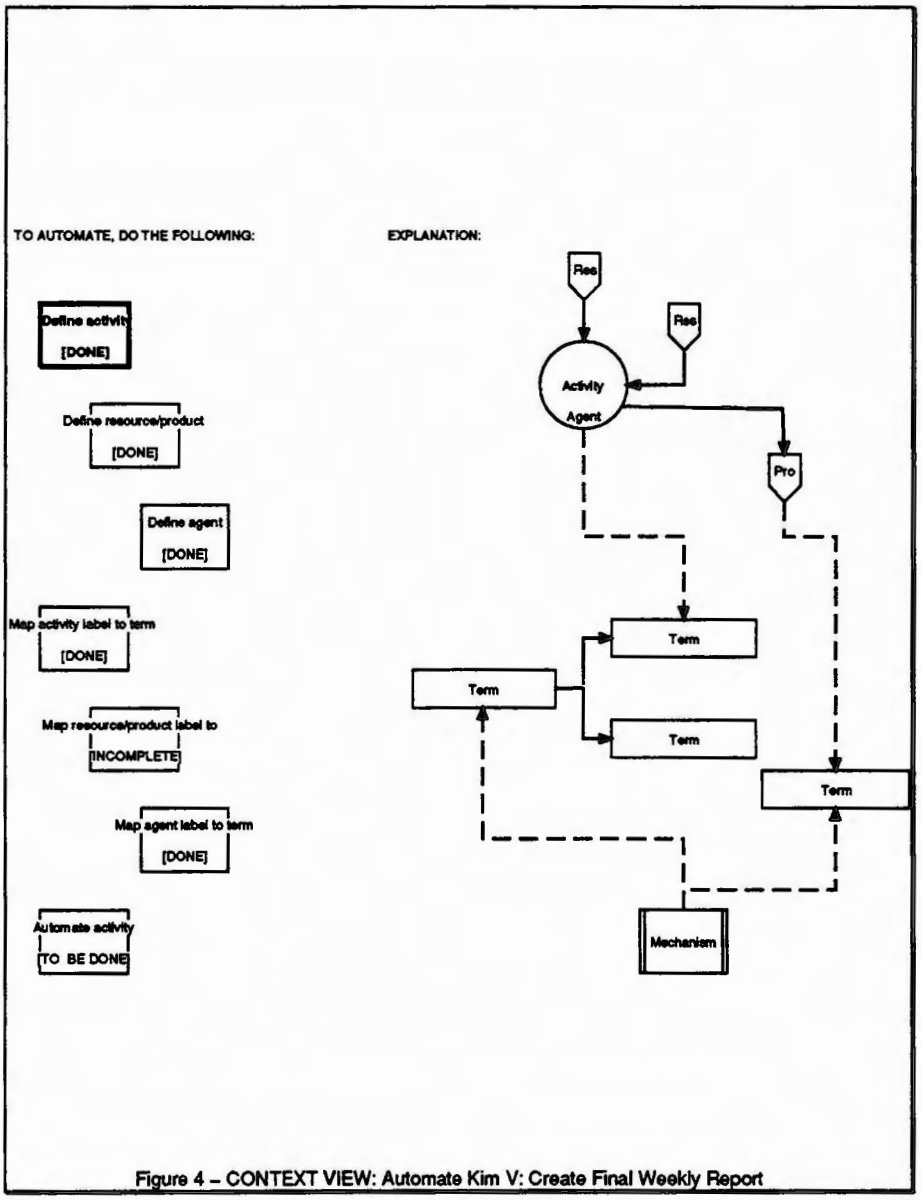


Figure 4 – CONTEXT VIEW: Automate Kim V: Create Final Weekly Report

That interface view contains status and context information on the left and an explanation of how the Active Glossary works on the right. The Active Glossary exploits terms, organized in a glossary, that mediate between an analyst's perspective and a programmer's perspective. The analyst uses this "design language" to make consistent the different individual descriptions of the

workplace. The design language contains the terms the analyst would use to indicate that two different labels used by different workers are really the same, or that the same label used by different workers are really different. Analysts build the glossary by entering their own terms, relating them to each other as they like. This is represented by the middle set of boxes on the right of Figure 4. When describing a specific activity from the workplace, the analyst maps the labels of the individual workers to the terms in the glossary. That mapping is represented by the dotted lines from the activity and product labels to the terms in the glossary. In addition to mapping labels used by the workers in the workplace to the design language, the analyst can introduce abstractions, refinements, and synonyms into the glossary, adding structure to the terms. In Figure 4, that structure is represented by the solid lines between the terms in the glossary. The set of labels that defines an activity's context consists of labels denoting the activity, resources, products, and agents. Each label from an activity's context is mapped to a term. That is, each activity is defined by a set of terms.

The design language is also used by programmers to describe the programming constructs they create in order to automate some of the activities in the workplace as defined by the analyst. In Figure 4, that mapping is indicated by the dotted lines from mechanisms to the terms in the glossary.

Once those links have been defined, the Active Glossary finds the best overlap between the terms defined for the activity and the terms defined for the programming constructs. It then suggests the programming constructs found as a means to automate the activity.

The interface view in Figure 4 indicates that activities, agents, products and resources have already been defined. Clicking on the "Define Activity" box provides the analyst with a summary of the information that was defined for the "Kim V: Create Final Weekly Report" activity (the same summary as shown in Figure 3). The interface view in Figure 4 further indicates that not all of the labels relevant to the activity's context have been defined with the terms in the glossary. The definition of the resource and product labels is "INCOMPLETE". By clicking on that box, the analyst can now define those labels.

2.2.2. Mapping Task Activities to the Glossary

The resource label "Weekly Report" still needs to be defined. Figure 5 shows that the Active Glossary suggests "Weekly Hours Report" as the term for "Weekly Report". It could make that suggestion because the label "Weekly Report" has been defined as "Weekly Hours Report" elsewhere in the workplace description.

This demonstrates one of the "active" components of the glossary. The Active Glossary uses simple heuristics to only display a subset of the glossary's terms. The heuristics determine which of the terms in the glossary are relevant to the current label. The heuristics used at this time are:

- If the current label is used elsewhere in the workplace description and that label has been defined by a term in the glossary, suggest to the knowledge engineer to use the same term for the current label.

- If the current label is not defined, but some of the labels from the current label's context are defined, display the terms that define the labels whose contexts contain the defined labels from the current label's context. The current label's context is defined by the labels that are close to the current label. For example, the context for a label denoting a workplace activity is defined by the labels denoting the activity's resources, products, and agents.

- If none of the heuristics apply, display the top-level terms of the structured vocabulary.

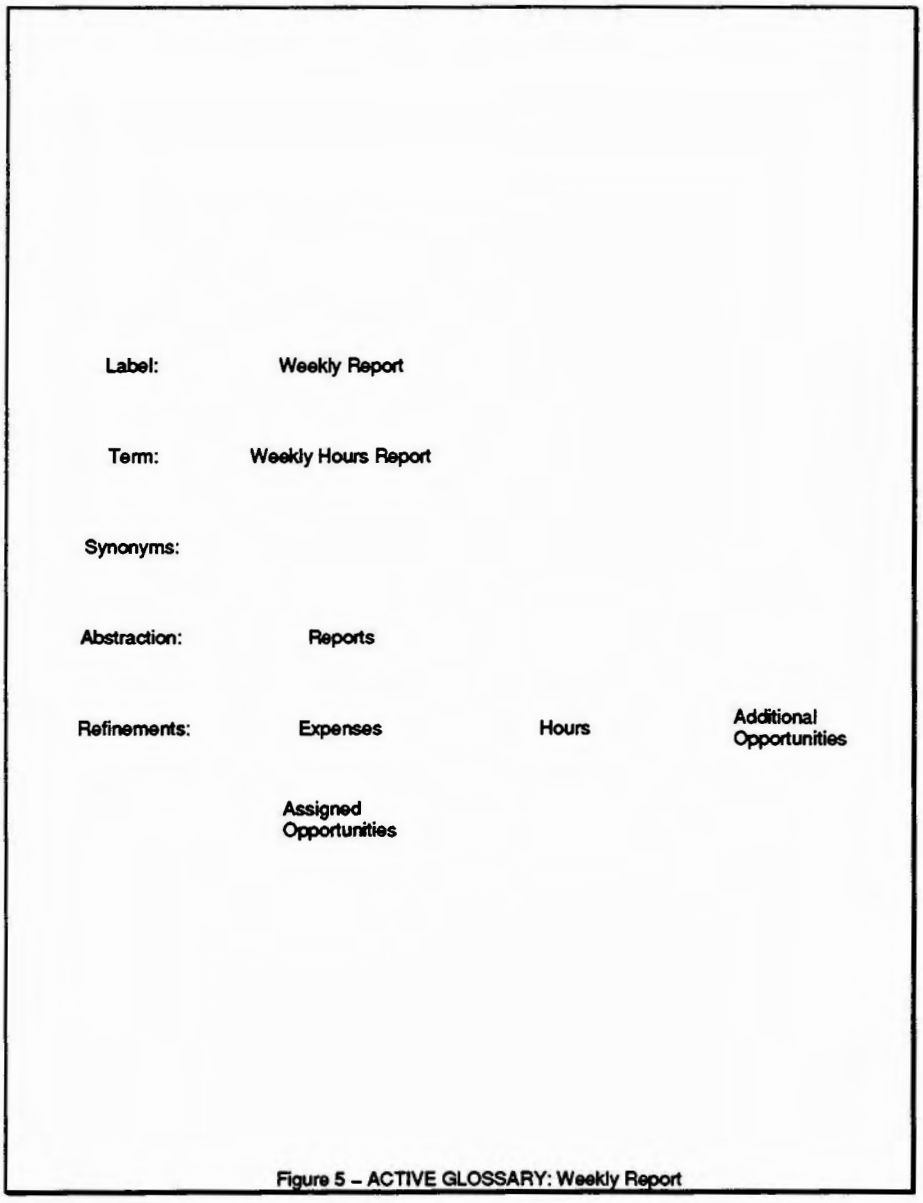


Figure 5 – ACTIVE GLOSSARY: Weekly Report

Continuing our example, the Active Glossary further indicates in Figure 5 the abstractions ("Reports") and refinements ("Expenses", "Hours", "Additional Opportunities", and "Assigned Opportunities") that have been defined in the glossary for the "Weekly Hours Report" term.

This demonstrates another "active" component of the Active Glossary. Terms are related to other terms in the vocabulary. The interrelations between terms divide the vocabulary into multiple tree structures. The Active Glossary exploits the structure when suggesting programming constructs that might automate an activity from the workplace description. The semantics of the links are not defined in the current version of the glossary. The structures that were created by our users are "type-of" and "part-of" relationships. By analyzing relationships that our users define, we hope to better understand the kinds of links that are important for a taxonomy. We will then introduce semantics for the links between terms.

The Active Glossary further assists a user in understanding a term in its vocabulary by presenting different summaries of the relevant terms. Figure 5 showed a textual view of the glossary. Other

views include an alphabetized list of the terms that can be reached if a user would follow the links from the current term to its subterms. Our example now demonstrates a third way to present the vocabulary to the analyst. Figure 6 shows a graphical view describing the term "Reports". This graphical view results from the analyst's decision to examine the glossary in more detail to better understand the term "Weekly Hours Report".

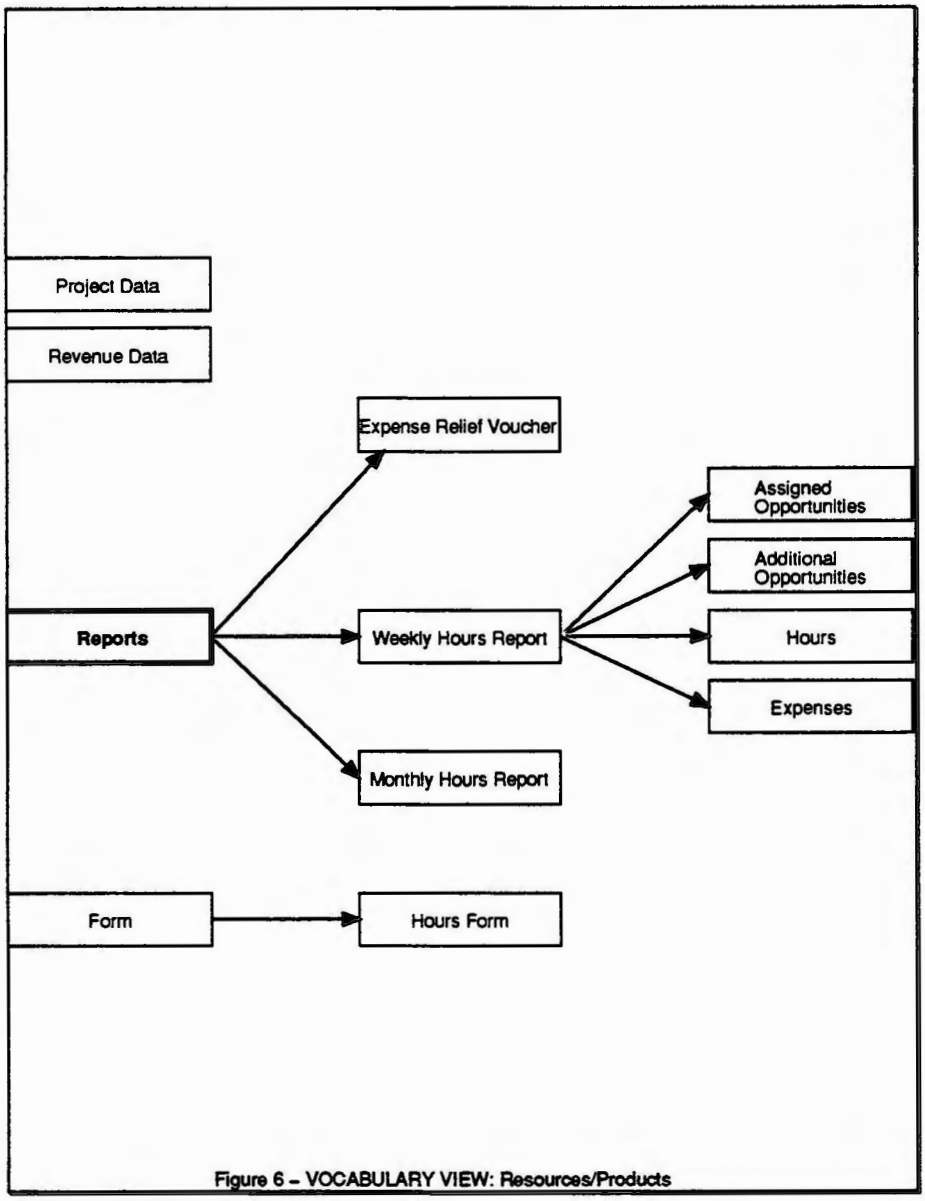


Figure 6 – VOCABULARY VIEW: Resources/Products

Not shown in this example is another "active" feature of the Active Glossary. The glossary maintains the links a user created when defining a label with a term from the vocabulary. A user can at any time examine how a term was used in the workplace description. The glossary will present the labels that are defined by the term in the context of their use, that is, it displays the views that contain the labels. This feature allows a user to understand how the terms in the glossary were used in the past and, thus, to use the term in the same way, maintaining consistency throughout the workplace description.

When users do not find a term in the glossary that defines the current label, the vocabulary can be extended simply by selecting the term icon, labeling it, and connecting it into the existing term structure. The Active Glossary has some extensive functionality that allows the user to automatically define labels in a workplace description with the terms in the glossary. For example, one mouse-click can update an entire workplace description with the vocabulary. To achieve that, the Active Glossary uses heuristics like the ones described earlier. If none of the heuristics apply, the label is put into the "unclassified" category of the glossary. The analyst must then integrate those terms into the glossary structure.

In our example, the analyst now agrees with the Active Glossary's suggestion. With the labels and terms defined, the user -- worker, analyst, or programmer -- can now ask for programming constructs that might automate the "Kim V: Create Final Weekly Report" activity, by selecting the box on the bottom in Figure 4 ("Automate Activity").

2.2.3. Configuring Mechanisms

The Active Glossary finds the best overlap between the set of terms defined for the activity and the set of terms defined for the programming constructs. If an activity term is the same than a programming construct term, this counts as a perfect match. If no perfect match is found, the Active Glossary tries to find a match for a more abstract term. If a match is found, this counts as an abstract match. If no match is found, the Active Glossary tries the refinements terms. If a match is found it includes all programming constructs on the refinement level in its suggestion. The Active Glossary does not configure programs to work together, it merely suggests which programs should be used together. It is up to the analyst and programmer to integrate workplace activities and programs.

In our example, clicking on "Automate Activity" (Figure 4) results in the list of programming constructs shown below in Figure 7.

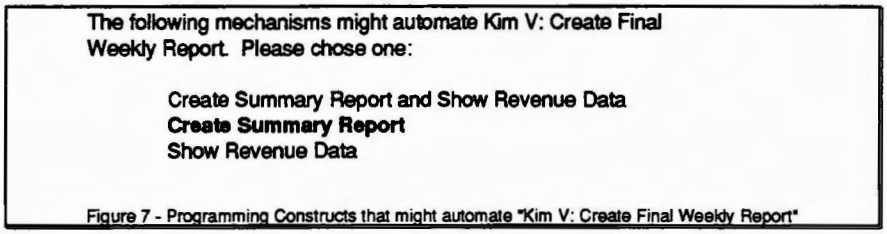


Figure 7 - Programming Constructs that might automate "Kim V: Create Final Weekly Report"

The analyst chooses "Create Summary Report". This is not a perfect match. Some of the activity labels do not overlap with the programming construct's label. The Active Glossary indicates this as shown in Figure 8.

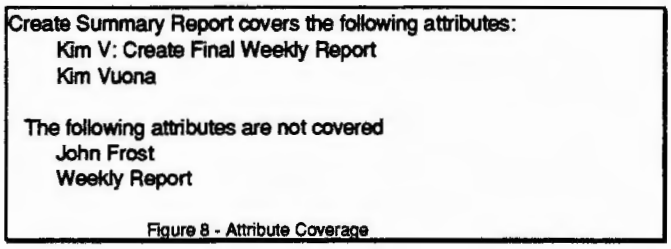


Figure 8 - Attribute Coverage

This can be used as a high level specification for a programmer to refine an existing programming construct or define a new one automating all of the activity. In our example, the analyst decides to use the programming construct. This information is passed on to the Bum tool.

2.3. Populating the Knowledge Base

Burn manages the knowledge-acquisition tools for our mechanisms. A knowledge-acquisition tool instantiates a reusable mechanism with the knowledge required to implement the current activity. Knowledge acquisition research has focused more on this part of the solution-development life cycle than on any other part. We have shown previously how the Spark tool shares knowledge with our knowledge acquisition tools, providing integration of knowledge [Marques et al. 1992]. In our example, the knowledge that instantiates the "Create Summary Report" mechanism includes a definition of the format or the report, the information that goes into the report, and strategies to elicit that information.

2.4. Integrating New Support into the Workplace

Once the "Create Summary Report" mechanism is instantiated, The Active Glossary creates a new view detailing the activity "Kim V: Create Final Weekly Report" and inserts the selected programming construct as an activity. When any new activities, agents, or other support are added to a workplace, they must fit into the other activities in the workplace. In our example, the new automated agent for "Kim V." must be told when to start, where to get its resources, and where to send its results. Introduction of a new agent usually requires some change to other activities. Using the information supplied by the Active Glossary, the analyst integrates the new automated agent with other manual activities to cover the activity labels that are not addressed by the selected programming construct. This results in the view detailing the "Kim V: Create Final Weekly Report" activity shown in Figure 9.

In Figure 9, the newly created automated agent has been added for the activity "Create Summary Report". The agent for the other two activities shown ("Add Information to Hfx", "Update the report or send it?") is human ("Kim V."). A workflow controller sharing the knowledge acquired during the workplace analysis will send a message to Kim V. when those activities are ready to be started, and Kim V. acknowledges completion when they have finished. The diamond-shaped activity is a choice, and appears on Kim V's workstation as a selection box. If she chooses 'Update report", then the automated agent for "Report Overview of Resource Allocation" is initiated, and, upon completion, the choice task is posted to Kim V. again. The small trapezoids at the top and bottom of the figure are the links to the more general view of the context of "Create Final Weekly Report", shown in Figure 2.

3. Some Data

The data evaluating the Active Glossary is mainly anecdotal. Currently, it is being used by 2 people -- an analyst (Spark, Burn, FireFighter effort) and a programmer (Documentation effort). As their efforts are ongoing, the following results are preliminary.

Spark, Burn, FireFighter effort: An analyst is using Spark and the Active Glossary to describe a workplace within Digital's AITC organization. She is an independent consultant not associated with Digital. Currently, her workplace description consists of 120 views with an average of 7 symbols each. The glossary consists of 117 terms organized in 15 views. It took approximately 2 person-days to create the glossary from the workplace description. Our analyst is a non-programmer. She used the glossary functionality to create a glossary from her workplace description, that is, after she created an initial description of the workplace ("passive" functionality). She found the current functionality quite cumbersome to use and we are working with her to improve it. For example, the context views (see Figure 4) and summary views (see Figure 3) are a result of her feedback to us. She will now start to use the Active Glossary while evolving the workplace description and we will gather data on the useability of the glossary's "active" functionality.

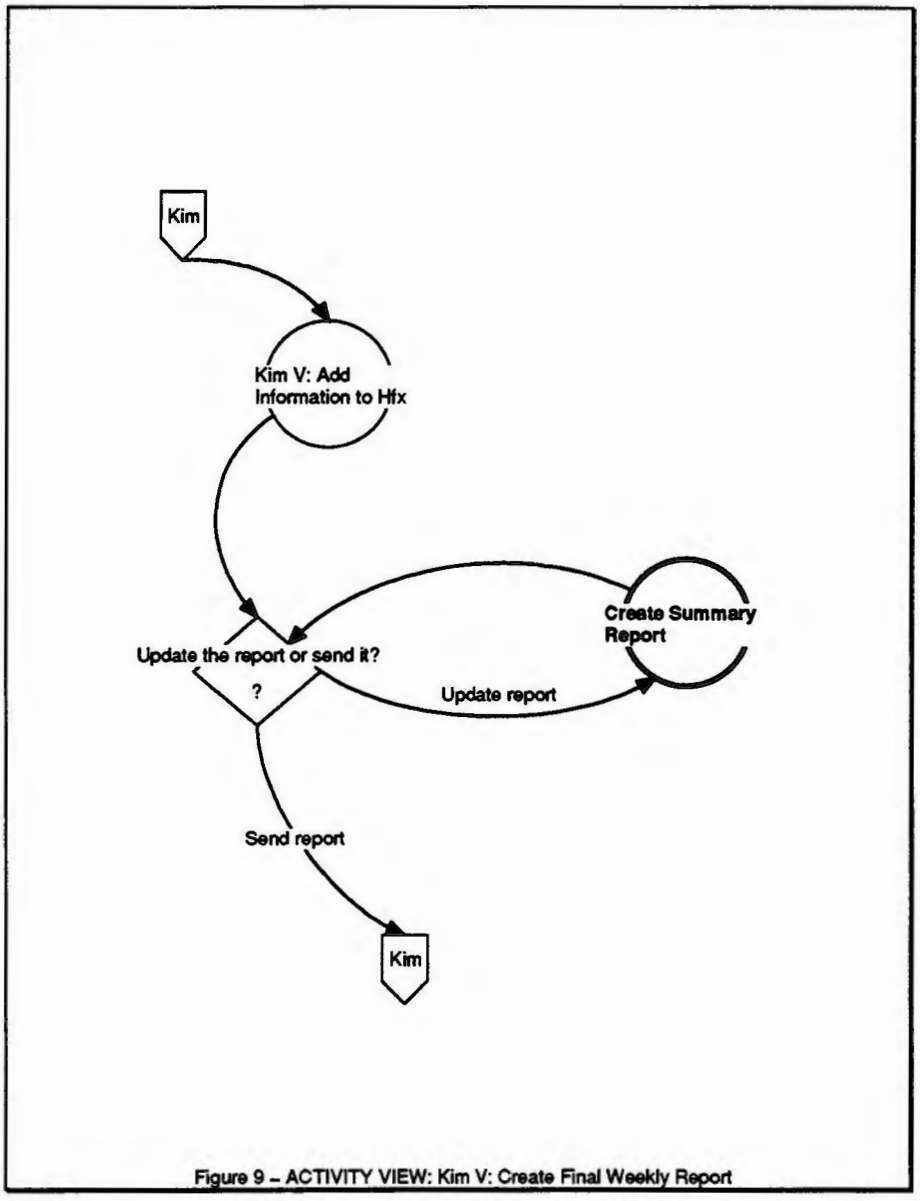


Figure 9 – ACTIVITY VIEW: Kim V: Create Final Weekly Report

Documentation effort: A programmer, who is not associated with our group at Digital is using the Active Glossary to document the design of a CASE tool. Currently, his description consists of 9 views with an average of 16 symbols each. The glossary consists of 53 terms organized in 12 views. It took approximately 1/2 person-days to create the glossary. In contrast to the analyst, the programmer was intimately familiar with the graphics interface used by all the Spark, Burn, FireFighter tools and found it quite easy to create and use the glossary. Once the documentation is complete, it will be introduced into the development group as an "active" documentation medium to assist in introducing new members to the code and identifying where to change the source code to remedy errors. We will monitor the use of the tool and report the results.

4. Related Efforts

The Active Glossary is a tool supporting the Spark, Burn, FireFighter knowledge-engineering environment. The Spark, Burn, FireFighter project is only one approach working on providing assistance to knowledge engineers throughout the life cycle of a knowledge-based system.

Traditional software-engineering frameworks such as the Yourdon method provide a set of tools to create a description of a workplace, design an automated solution, and implement that solution [Ward 1985]. Tools include context diagrams, data flow diagrams, structure charts, state transition tables, and data dictionaries. This approach -- and all other solution-development approaches -- emphasizes that a workplace analysis collects knowledge about a workplace that must be used during all other phases of designing and implementing automation support. The more successful this phase is, the less effort is needed to produce an automated solution.

The DIDS project [Balkany et al. 1991, Runkel et al. 1992] takes an approach similar to that of Spark, Burn, FireFighter. DIDS is developing a graphics-based programming environment for developing configuration-design tools. The environment provides a tool to build a problem-solving method using mechanisms. A knowledge-acquisition tool is then automatically generated. The mechanisms in DIDS are the result of an extensive and ongoing analysis of the tasks that compose configuration design. The mechanisms have the features of reusability and combinability, which makes programming easier. Reusability reduces the amount of programming needed to build a system. Combinability ensures that mechanisms can be pieced together. The knowledge-acquisition tools produced by DIDS are specialized to each problem-solving method developed.

PROTEGE II [Tu 1991] is a framework for knowledge engineers to generate knowledge-acquisition tools. It includes an architecture to assemble problem-solving methods from lower-level building blocks. This work emphasizes a uniform view of problem solving at different levels of granularity, an explicit data model to construct new data types from predefined data types, and the inclusion of domain-dependent control information. The building blocks are similar in scope to Spark, Burn, FireFighter's and to DIDS's mechanisms.

The KADS and KADS II projects [Wielinga, 1992] are developing a library of building blocks (knowledge sources in their terminology) that appear to be of a scope similar to that of Spark, Burn, FireFighter's, PROTEGE's and DIDS's mechanisms. The researchers have built a formal framework of interrelated models to guide the process of building applications. The biggest difference among these systems is that the KADS framework is not intended to be operational. It defines a methodology aimed at helping a programmer to design an application.

5. Requirements for Knowledge-Acquisition Tools that Take Integration Seriously

Our Spark, Burn, and FireFighter framework and those described in Section 4 indicate that taking integration seriously means the following things for knowledge-acquisition tools:

- Knowledge acquisition happens throughout the solution-development life cycle (see Section 1). Traditionally, knowledge-acquisition research has focused more on supporting the population of a predefined knowledge base than on other parts of the life cycle. Analyzing the workplace, designing a problem-solving approach and knowledge base, and introducing new support into the workplace are life-cycle phases that are equally important for knowledge acquisition. For example, the Active Glossary facilitates the creation of a shared vocabulary. This vocabulary supports a knowledge engineer with analyzing the workplace and designing a problem-solving approach.

- Knowledge-acquisition tools need to know the kind of information that is shared across phases of the life cycle. They need to know what pieces of information are relevant for them, how they are represented, and where to find the relevant information. For example, the Active Glossary presupposes a workplace description and a mechanism description. This information is provided by the Spark tool. Both tools share the same representation for

workplace and mechanism descriptions. The descriptions are stored in the same database. An alternative approach is to allow different representations for different tools. The different representations are then "translated" into and from a common representation. This approach is explored by other tools in the Spark, Burn, FireFighter framework.

- Knowledge-acquisition tools must support knowledge engineers to easily move back and forth between integrated tools. That is, they need to provide the appropriate information at the appropriate time in the appropriate form. For example, the Active Glossary provides interface views that contain status and context information (see Section 2.2.1). Context information presents the tasks a knowledge engineer must perform given a goal to be accomplished. The tasks can be initiated from the context information. Status information describes whether a task has been accomplished, is not yet completed, or still needs to be done. Though they are separate, loosely-coupled tools, the context and status information integrates the Active Glossary and Spark so that the knowledge engineer will switch among the tools without knowing the difference.

- Knowledge-acquisition tools need to actively support integration. Different tools have been designed to operate on different bodies of knowledge. Mapping the different bodies of knowledge onto each other becomes a major issue. For example, one of the tasks a knowledge engineer must perform is to identify programming constructs that can be used to automate some of the activities in the workplace. To do this, the knowledge about the workplace (the analyst's knowledge) must be shared with knowledge of how to construct application programs (the programmer's knowledge). Just providing the different bodies of knowledge and leaving it up to the knowledge engineer to make the connection is not helpful. For example, the Active Glossary assists a knowledge engineer with creating an ontology. But only providing an ontology would leave it up to the knowledge engineer to map the ontology to Spark's workplace and mechanism descriptions. This would make a knowledge engineer's task even more difficult. Therefore, the Active Glossary guides knowledge engineers in mapping their understanding and description of a workplace and mechanisms onto terms in the glossary by illustrating how the terms were used previously (see Section 2.2.2).

- Knowledge-acquisition tools that know the kind of information that is shared across phases of the life cycle, that support knowledge engineers to easily move back and forth between integrated tools, and that actively support integration are methodology dependent. For example, the Active Glossary has its place within the Spark, Burn, FireFighter framework. It takes advantage of a representation that is common to all tools in the framework; its interface view exploits the life-cycle phases supported by the framework; and it knows how to map workplace descriptions and mechanism descriptions onto its ontology.

Our approach to building an environment to assist in the full life-cycle of adding support to a workplace is to build knowledge acquisition tools specialized to seek and represent the knowledge they need for their specific tasks within the life-cycle. These tools must then put that knowledge in a form and place where other tools can share it. All tools must also actively seek and use knowledge gathered by other tools, as illustrated by the Active Glossary. In this view, each tool places constraints on each other tool, to ensure that knowledge is acquired when the user will best be able to provide it, and that as rich a context as possible is provided for the person to understand what is being asked. Finally, each tool must assist with the introduction of new support into the existing workplace, that is, each tool must facilitate change. This assistance helps ensure that the new support will be useful and used, thus, allowing the evolution of effective application programs.

Acknowledgments

We would be remiss if we did not mention our co-workers in this project: Glee Cameron, Geoffrey Dallemagne, Charlie Reed, Yuri Tijerino, Tina Whitney, and Gregg Yost are or were members of the group and made significant contributions.

References

Balkany 1991: A. Balkany, B. Birmingham, and I. Tommelein. A Knowledge-Level Analysis of Several Design Tools. *2nd International Conference on AI in Design.* Butterworth Scientific Ltd, 1992.

Birmingham 1988: W. Birmingham. Automated knowledge acquisition for a computer hardware synthesis system. In *Proceedings of the 3rd Knowledge Acquisition for Knowledge-based Systems Workshop.* Banff, Canada, 1988.

Boose 1984: J. Boose. Personal construct theory and the transfer of human expertise. In *Proceedings of the 4th National Conference on Artificial Intelligence.* Austin, Texas, 1984.

Boose 1989: J. Boose. A Survey of Knowledge-Acquisition Techniques and Tools. *Knowledge Acquisition.* Vol.1, No.1, pp. 3-38, 1989.

Chandrasekaran 1990: B. Chandrasekaran. Design Problem Solving: A Task Analysis. *AI Magazine.* ():54-71, 1990.

Clancey 1985: W. Clancey. Classification problem solving. In *Proceedings of the 4th National Conference on Artificial Intelligence.* Austin, Texas, 1984.

Dallemagne 1992: Dallemagne, G., Klinker, G., Marques, D., McDermott, J., Tung, D. Making Application Programming More Worthwhile. To appear in *Knowledge Engineering and Cognition*, Schmalhofer, F., Strube, G., Wetter, Th., eds., Springer Verlag. 1992

Diederich 1987: J. Diederich, I. Ruhmann, and M. May. Kriton: a knowledge acquisition tool for expert systems. *International Journal of Man-Machine Studies.* 6(1): 29 -- 40, 1987.

Eshelman 1987: L. Eshelman, D. Ehret, J. McDermott, and M. Tan. Mole: a tenacious knowledge acquisition tool. *International Journal of Man-Machine Studies.* 26(1):41 -- 54, 1987.

Friedland 1989: P.E. Friedland and Y. Iwasaki. The concept and implementation of skeletal plans. *Journal of Automated Reasoning.* 1(2):161 -- 208, 1985.

Klinker 1989: G. Klinker, Casey Boyd, David Dong, Jon Maiman, John McDermott, and Robert Schnelbach. Building expert systems with knack. *Knowledge Acquisition.* 1(3):299 -- 320, 1989.

Klinker 1988: G. Klinker. Knack: sample-driven knowledge acquisition for reporting systems. In *S. Marcus (Ed)., Automating Knowledge Acquisition for Expert Systems.*, Kluwer Academic Publishers, Boston, August 1988.

Linster 1989: M. Linster. Towards a second generation knowledge-acquisition tool. *Knowledge Acquisition.* 1(2):163 -- 184, 1989.

Marcus 1988: S. Marcus. Salt: a knowledge-acquisition tool for propose-and-revise systems. In *S. Marcus (Ed)., Automating Knowledge Acquisition for Expert Systems.*, Kluwer Academic Publishers, August 1988.

Marques 1992: D. Marques, G. Dallemagne, G. Klinker, J. McDermott, and D. Tung. Easy Programming: Empowering People to Build Their Own Applications. To appear in *IEEE Expert.* 1992.

Marques 1988: D. Marques, A. Latto, and J. McDermott. A comparison of case based reasoning with few large vs many small features. In *Proceedings of the Workshop on Case Based Reasoning.* St. Paul, August 1988.

McDermott 1988: J. McDermott. Preliminary steps toward a taxonomy of problem-solving methods. In *S. Marcus (Ed)., Automating Knowledge Acquisition for Expert Systems.* Kluwer Academic Publishers, August 1988.

Musen 1989: M. Musen. Automated Generation of Model-Based Knowledge Acquisition Tools. Pitman, 1989.

Newell 1982: A. Newell. The Knowledge Level. *Artificial Intelligence.* 18():87-127, 1982.

Runkel 1992: J.T. Runkel, B. Birmingham, T.P Darr, B.R. Maxim, and I.D. Tommelein. Domain Independent Design System: Environment for Rapid Development of Configuration Design Systems. *2nd International Conference on AI in Design.* Butterworth Scientific Ltd, 1992.

Shaw 1987: M.L.G. Shaw and B.R. Gaines. Techniques for knowledge acquisition and transfer. *International Journal of Man-Machine Studies.* 27(3):251 -- 280, 1987.

Steels 1990: L. Steels. Components of Expertise. *AI Magazine,* 11(2), 1990.

Tu 1991: S. Tu, Y. Shahar, J. Dawes, J. Winkles, A. Puerta, and M. Musen. A Problem-Solving Model for Episodic Skeletal-Plan Refinement. *Proceedings of the 6th Knowledge Acquisition for Knowledge-based Systems Workshop.* Banff, Canada, 1991.

Ward 1985: Ward, P.T., and Mellor, S.J. Structured Development for Real-Time Systems. Prentice-Hall, 1985.

Westphal 1989: C. Westphal, and K. McGraw (Eds). Special Issue on Knowledge Acquisition. *ACM SIGART.* (108), 1989.

Wielinga 1991: B. Wielinga, A.Th. Schreiber, and J.A. Breuker. KADS: A Modeling Approach to Knowledge Engineering. ESPRIT Project P5248 KADS-II/T1.1PP/UvA/008/1.0, University of Amsterdam, 1991.

Zuboff 1988: S. Zuboff. In the Age of the Smart Machine: The Future of Work and Power. Basic Books Inc., New York, NY, 1988.

INTEGRATING TECHNIQUES FOR CONCEPTUAL MODELING

Timothy C. Lethbridge
Doug Skuce

Department of Computer Science
University of Ottawa
Ottawa, Ontario, Canada K1N 6N5
(613) 564-8155 tcl@csi.uottawa.ca
(613) 564-5418 doug@csi.uottawa.ca

ABSTRACT

We describe seven knowledge engineering projects with which we have been involved. In these projects the knowledge represented is largely descriptive, has a significant informal content, and typically is creative as opposed to the being result of expertise-capture. We then compare eight major semi-automated techniques we have found most useful in the projects The techniques are: brainstorming, interpretive structural modeling, text scanning, conceptual diagraming, conceptual outlining, table completion, systematic critiquing and extended inference. Many of these techniques are widely known, but we have tailored them for our purposes. We suggest criteria for analysing the techniques and describe the techniques in terms of the these criteria. We also indicate how our knowledge management system, CODE4, has been developed with these techniques in mind in order to effectively support our knowledge engineering projects.

1. INTRODUCTION

Our goal is to facilitate the conceptual modeling of a domain using a flexible knowledge representation schema and supporting software. To facilitate this we have built several knowledge management systems, the latest one being CODE4[1] (Skuce and Lethbridge 1992)(Skuce 1992c).

In the rest of this introduction we describe the type of knowledge engineering we do. In section 2 we describe eight major techniques and list some properties and advantages of each technique. We analyse several of our knowledge engineering projects to see what techniques were used and with what results. We describe how CODE4 has evolved to support these techniques in a highly integrated way, and where we see room for further automated assistance. Finally, we present some some heuristics that indicate what techniques seem most appropriate to what kind of project.

1.1 Our flavor of knowledge engineering

As emphasized by Bradshaw et al (Bradshaw, Chapman et al. 1992) it is becoming generally accepted that many aspects of knowledge engineering primarily involve

---

1 CODE stands for Conceptually Oriented Design/Description Environment. The current version is CODE4 which in a complete redesign of CODE2. The knowledge engineering projects described in this paper have used both versions. CODE3 was an experiment that was never used for practical knowledge engineering.

constructive modeling. The old emphasis on *extracting expertise* is often a secondary focus, and may not be present at all.

In our work, we also find that we primarily need to represent the following types of knowledge (which are typically highly correlated):

a) Declarative (i.e. non-procedural) knowledge.

b) Descriptive properties of things (i.e. not rules).

CODE4 and its predecessors are primarily designed, although not restricted, to handle these types of knowledge.

CODE4 reflects our increased understanding of the importance of:

- The knowledge representation: its comprehensiveness and expressiveness.
- User interface design for knowledge management systems.
- General knowledge management[2] techniques.

In this paper we focus on the techniques; this necessitates describing a few aspects of the user interface. The following paragraphs provide a few points about the knowledge representation so that the reader can better understand the rest of the paper:

The units of knowledge in CODE4 are termed concepts. These are categorized into: type concepts, instance concepts, terms, predicates, statements, rules etc. Concepts are the machine's model of anything it can reason about. Analogously we can think about concepts in the mind as representing anything the mind can think about.

CODE4 has a number of advanced knowledge representation features (Lethbridge 1991a). Of particular note is its ability to represent concepts on a formal-informal spectrum (Lethbridge 1991b), (Lethbridge and Skuce 1992). Informal knowledge is usually in the form of text strings that cannot be immediately interpreted by the system, and relies on interpretation by the user of the knowledge (people or other systems). Formal knowledge is tightly connected to other knowledge with links that have a well-defined semantics.

1.2 Recent knowledge engineering projects

The following paragraphs describe some of the major projects in which we have been involved, using CODE2 and CODE4. The list is not exhaustive: we have built numerous other small knowledge bases, and various other individuals and organizations use CODE for their own purposes, e.g. (Bradshaw, Holm et al. 1992)

- **The Bell-Northern Research Telos project**

  This project (Skuce 1992b) lasted 18 months and consisted of building a knowledge base about Telos, a complex software system under development. The effort included reverse engineering Telos as well as assisting in the conceptual design of new features. Two knowledge engineers and several software engineers and managers were involved.

---

2 By *knowledge management*, we include knowledge acquisition, the refinement of the knowledge, and making the knowledge available for use (i.e. as a server). In this paper we focus on the first two of these aspects.

- **The Telebrain project**

  In this project we designed a hypothetical 'invention', a new kind of telephone system. The focus was on allowing the knowledge engineer to be as creative as possible (as he or she might be if using traditional tools such as paper and pencil) while gaining the power of a knowledge representation system.

- **The Ontology project**

  [3]This on-going project (Skuce and Monarch 1990) is attempting to identify a useful set of general concepts for the top of an 'isa' hierarchy, based on linguistic research. The objective is to achieve a conceptually elegant ontology that as many people as possible can agree on, and use as a basis for exchanging knowledge bases.

  So far, we have developed a number of iterations of the ontology. Each time we have refined our thinking and incorporated the ideas of others (Lenat and Guha 1990), (Miller 1990).

- **The CODE self-description project**

  Several times during the development of CODE2, CODE3 and CODE4 we created knowledge bases describing our conceptual understanding of the system under design.

- **The Cogniterm project**

  This on-going project (Meyer, Skuce et al. 1992) is exploring how a knowledge management tool can assist the terminologist. Much of the focus is on defining terms and describing how they are used.

  The main knowledge base contains descriptions of several hundred concepts in the domain of 'optical storage media'.

- **The Smalltalk description project**

  As a new tactic in our strategy of using CODE for software development, Iisaka (Iisaka 1992) has created a knowledge base about the Smalltalk language (in which CODE2 and CODE4 are implemented). CODE4 has also been extended so that the knowledge base can make direct reference to Smalltalk objects.

2. HIGH LEVEL CONCEPTUAL MODELLING TECHNIQUES

Figure 1 is a 'knowledge flow diagram' showing the main techniques that we use. Knowledge entering by any of the techniques comes from the minds of people, from documents or from knowledge already existing in the knowledge base.

The subsection 2.1 describes some of the criteria by which we can categorize the techniques and subsection 2.2 describes the techniques in more detail. In section 2.4 we indicate how the various projects described above made use of the techniques.

---

3 The word 'ontology' is developing a variety of meanings. Here we use it in the sense of highly general concepts applicable to a wide variety of domains.

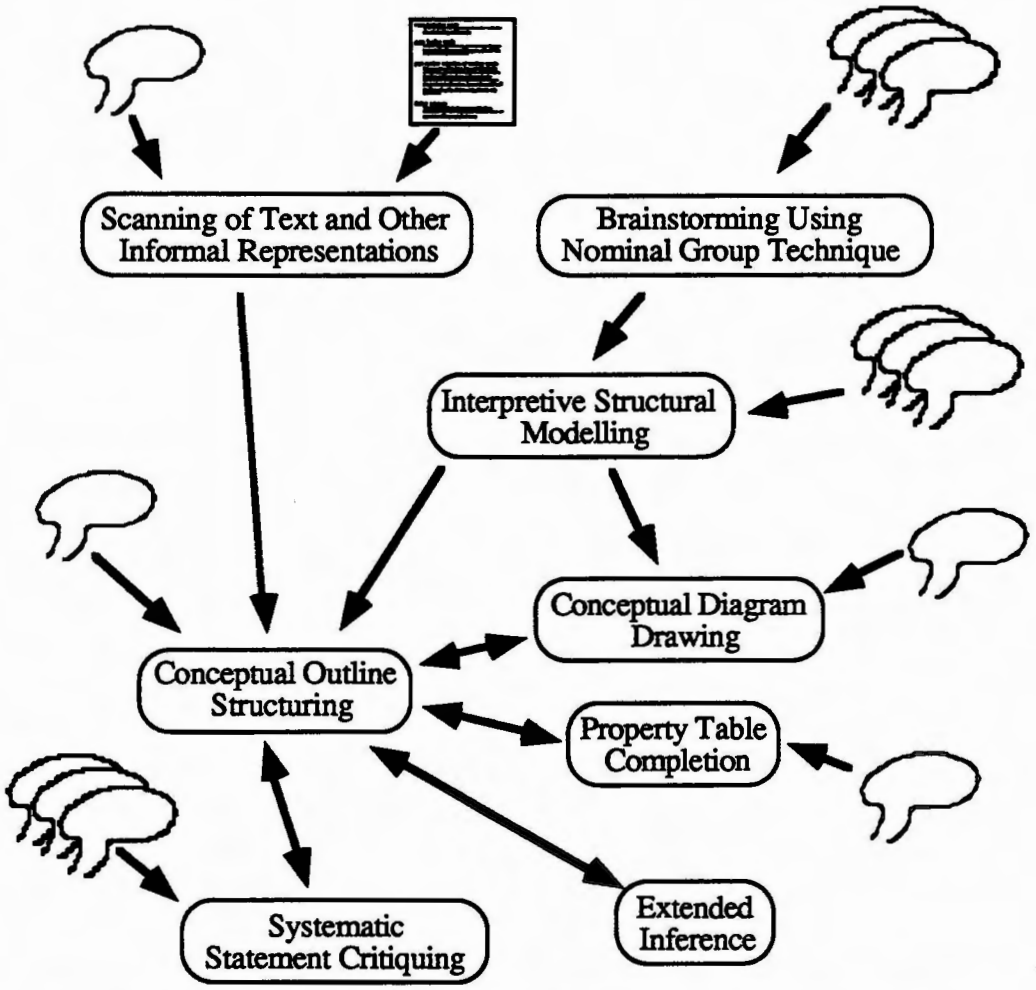


Figure 1: A knowledge flow diagram showing the various high-level knowledge acquisition techniques we use as knowledge is gradually formalized. The arrows represent the flow of knowledge between techniques (they do not indicate time sequencing since, for example, brainstorming may be used several times to expand the knowledge). Note that conceptual outline structuring plays a central role: we have found this to be the most generally useful technique.

2.1 General questions to ask about the techniques

The techniques in figure 1 all help build up a knowledge base, but they have very different properties. In particular we can ask the following types of question of each technique:

- **Pre-existing knowledge**

  What knowledge needs to be present prior to the technique being used? What is the starting point for the technique?

  Most techniques require existing knowledge that they can refine or use in structuring other knowledge. For some techniques this input knowledge needs to be relatively formal, whereas for others it is preferable that it be in the form of informal natural language statements.

- **Type of knowledge that results**

  Is the resulting knowledge new, or is it a refinement of pre-existing knowledge?

  It has been said (Skuce 1992a) that there are three main activities that can be performed by users of a knowledge management system: 1) acquiring new knowledge (often involving the consensus of several people), 2) critiquing and refining existing knowledge, and 3) querying, extracting and referring to the knowledge (using the knowledge base as a server). The first two of these are the knowledge acquisition activities that concern us here, although 3 is a constant need during 1 and 2. Some of the techniques are primarily oriented towards new knowledge, whereas others are oriented towards refining knowledge.

  Is the resulting knowledge informal or is it tightly connected to existing knowledge in the knowledge base (i.e. is it formal)?

  Some of the techniques result in informal (typically natural-language) statements or loosely structured knowledge, while others increase the linkages between concepts, making the knowledge base more formal.

  Answers to the two questions above are correlated: Techniques that generate new knowledge tend to generate informal knowledge. Techniques that refine knowledge tend to make it more formal.

- **Creativity and synthesis**

  Is the user free to enter whatever comes into his or her head, or is he or she constrained to answer questions or follow a machine prescribed procedure? Who initiates the next action, the user or the machine?

  Two attributes of a machine-assisted technique can answer this question: modality and interruptability. The former applies to fine grained actions in the user interface, the latter applies to switching back and forth between techniques.

  If the user interface is *modal*, the user must respond to prompts. Most user interfaces have more modality than is strictly necessary: For some techniques it is possible and desirable to remove all modality (the user is in complete control, no keystroke or movement of the mouse is requested by the machine); other techniques naturally require a modal interface.

  If the technique is *interruptable*, then the user can switch to another technique, suspending what he or she is doing in the current technique. This is generally good, but can be to distractive for some techniques.

  We hypothesize that less modality and greater interruptability have the potential to stimulate *creativity*: The user is not constrained to a particular course of action, and this is better able to represent whatever thought come into her or his mind. Greater interruptability also helps boost *synthesis*, since the user can go looking for knowledge to combine.

  What feedback should the user get to stimulate further ideas?

  Some techniques are more oriented towards giving the user freedom to enter knowledge, whereas others focus on displaying the consequences of what is entered. Feedback can be of three forms:

  1) *Group participant feedback*: Other participants in the technique see what has been entered and respond

2) Computer responsive feedback: The computer displays the consequences of user editing actions (in some systems this is done modally, with the user being forced to respond to the feedback, however we find it best not to force responses even if the user has entered something 'wrong')

3) Computer navigational feedback: The computer displays knowledge structures as the user navigates, often these knowledge structures must be dynamically computed. Typically the user will navigate around the knowledge base after every edit. The consequences of previous edits then become clear. This type of feedback differs from the last in that the user must specifically request it (although the request may be implicit).

- **Group efficacy**

  Is this technique suitable for use by a group?

  Some techniques are naturally group techniques. For these we would want to build 'groupware' user interfaces where several people can interact in parallel on their own screens. Other techniques are naturally oriented to individual knowledge acquisition. Some techniques are better performed on one screen (no parallel input) but can be effectively performed with a group of people guiding an operator.

In the next subsection we answer the questions in the context of each technique. The answers are summarized in table 1. In table 2, we also answer some of the questions in the context of the major projects in which we used the techniques.

The above questions are certainly not the only ones we can pose of the techniques. For example it might be useful to ask how good each technique is at producing problem-free knowledge (perhaps using the eight criteria listed under the systematic statement critiquing technique in the next section). This is a topic for future research.

2.2 Techniques amenable to computer assisted conceptual modeling

In this section we describe in detail the eight techniques shown in figure 1. We answer the questions posed in the last subsection and describe some of our experiences with the techniques.

The common threads that ties all the techniques together are: 1) they have been found effective for the type of conceptual modelling described in section 1.1, and 2) they can usefully be packaged together in an integrated computer assisted knowledge acquisition system, as indeed we are currently doing in CODE4.

- ***Brainstorming using nominal group technique***

  Starting point: A *trigger question*, of which participants are to think of as many answers as possible. No other knowledge need exist. This is decided by the group or moderator; it gives an essential focus to the session.

  Result: New, largely informal knowledge: a list of concepts that are answers to the trigger question. The concepts are typically expressed merely as text strings, although graphical output (sketches) might also be possible. The resultant concepts are typically a mixture of type concepts, instance concepts and statements; however it is up to a later technique to actually categorize them as such and to take other steps to formalize them.

  Technique: This technique is typically manual (Delbecq and al 1975), but recently computerized brainstorming assistance tools have appeared. Participants are arranged in a cycle and attempt to think of answers to the trigger question. As soon as they think of a possible answer it is made available to the next participant in the cycle. Similarly, an-

swers from previous participants in the cycle stimulate new lines of thought, and hence answers.

Our insight is to treat the items that result from brainstorming as concepts in their own right. Such concepts would be very informally specified, but a system like CODE4 is capable of dealing with any level of formality. Later techniques would incrementally increase the formality of the knowledge. We envisage participants working at their own screens, with windows for input and for browsing of the ideas of others.

Interface modality: Fully non-modal. At any time, the participant is free to inject new answers, to browse previous answers etc.

Interruptability: Best not interrupted to use another technique because that would result in a participant effectively dropping out of the nominal group.

Feedback potential: High, but originating from the parallel efforts of others answering the trigger question. There is no opportunity for the system to comment on participants' input since it is too informal.

Group-efficacy: Requires a group.

Our experiences: In the BNR project we used this technique several times with no computer support, subsequently performing interpretive structural modelling (below). and then entering the results into a CODE2 knowledge base using conceptual outline structuring (see below).

We have also tried other less structured approaches to brainstorming, particularly conceptual outline structuring (below) with a large screen shared by a group. These approaches, while useful, do not generate as large an amount of new ideas and group participation as nominal group technique.

Example: When designing a subsequent version of CODE we held a brainstorming session with the trigger question, 'What features should we add in order to provide better support for requirements analysis?'. Some thirty ideas were generated by brainstorming. Many of these were not new, but some gave us fresh insight. The speed with which we could completely generate a more or less complete spectrum of ideas was particularly useful.

- ***Interpretive structural modelling (ISM)***

Starting point: A list of more-or-less related concepts. These may be the result of brainstorming or informal representation scanning.

Result: The starting concepts grouped, categorized or arranged in other hierarchies or arbitrary graphs according to some set of relations. Thus existing knowledge is refined and made more formal.

Technique: Historically, a manual technique (Warfield 1976), but computerized assistance exists as exemplified by BNR's PRISM tool (Saunders 1990). The procedure varies depending on the number of concepts and the type of structure to be built, but the main ingredients are a well-structured and rapid sequence of moderated discussions and votes.

Once again, our insight is to structure concepts directly using a knowledge based system. For example ISM can be used to classify concepts when building an isa hierarchy, or to structure conceptual relations such as temporal or importance graphs.

Interface modality: Largely modal. The group members respond to prompts that request them to vote.

Interruptability: Interruptable to perform brainstorming or some other technique, particularly when the group feels that the input concepts are inadequate to compose the structures being built.

Feedback potential: Typically low. It is possible to observe the structure being built, but this can distort the elicited knowledge. It is generally best to let the participants respond to the prompts by thinking about what they are being asked rather than attempting to look it up.

Group-efficacy: Most useful in a group, but can be performed individually.

Our experiences: We frequently perform this technique following the use of the nominal group technique to place some structure on the answers to the trigger question.

Example: After brainstorming for a list of concepts (representing features) we decided to categorize them. The first step was to take an arbitrary pair of concepts and ask the participants, 'Do these fall in the same category?' If there was disagreement a discussion was held. This was followed by a vote. This pairwise-voting procedure was repeated until the computer determined that it had sufficient information to categorize all concepts. After naming the categories we performed further ISM to arrange the concepts (including the new categories) in an 'is more important than' relation.

In this particular example, the categorization resulted in an informal 2-level isa hierarchy. Part of the informality was due to the fact that the concepts were still expressed as character strings. Additional informality resulted from the fact that the categories were often not really types or superconcepts of the input concepts. A purist knowledge engineer would in fact have been rather displeased at this initial attempt at creating an isa hierarchy. Nevertheless, *real knowledge was generated* and a knowledge engineer was subsequently able to rapidly clean up the knowledge. The categories provided a very useful breakdown of types of features, a breakdown that nobody would have individually thought of. Given the categories, it became possible to generate further knowledge by asking the question, 'What is missing from this partition?' during systematic statement critiquing (below).

- ***Scanning of text and other informal representations***

Starting point: A possibly-complex informal representation that cannot be fully parsed and compiled into a formal knowledge structure. Typically this is a natural language document or the description of a single concept using natural language. In some future implementation it might be possible to scan visual representations looking for patterns, icons etc.

Result: A series of new terms extracted from the input.

Technique: Much knowledge that we would like to capture in a knowledge base is often expressed in natural language documents. We believe that with current technology it is unreasonable for a knowledge-based system to attempt to 'understand' an entire text document.

There have been many attempts to extract terms out of documents *without* full natural-language understanding (Szpakowicz 1990). CODE2 can take a document and look for terms that are not contained in its lexicon. These are presented to the user as possible candidates for inclusion in the knowledge base (see figure 2). The user may accept the term (possibly declaring information such as part-of-speech), or reject it.

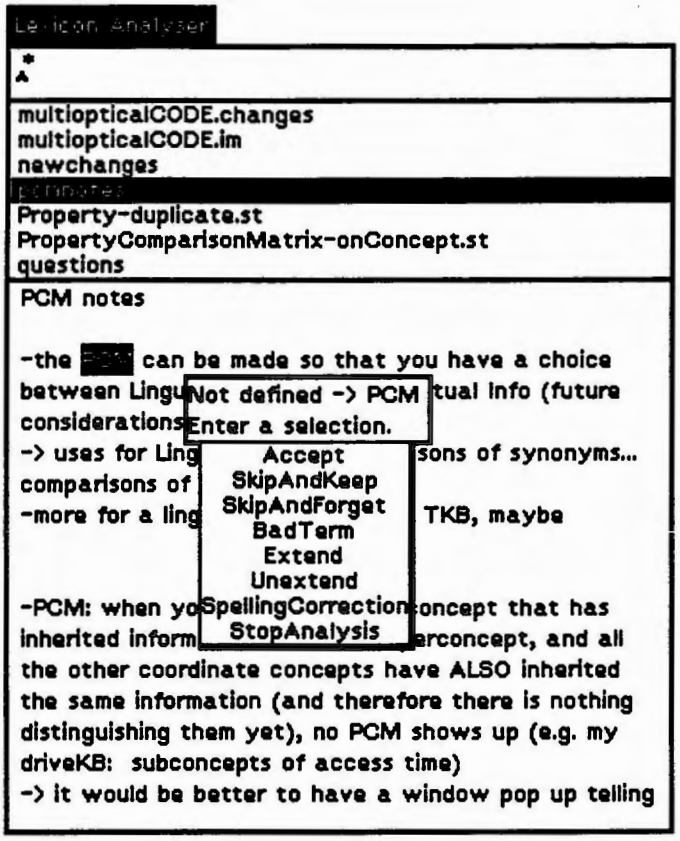


Figure 2: Text scanning of a natural-language document. The term 'PCM' has not been found in the Cogniterm knowledge base.

As an extension to the technique, the scanning process can pause on terms it already knows, presenting its current knowledge about the concept or concepts(senses) the term means. The user can then be prompted to confirm or declare the correct sense. As an future extension, built into a hypertext system, the system might tag the term in the input document with a pointer to the knowledge base.

Interface modality: Largely modal. The user is presented with a sequence of prompts requesting acceptance, rejection, confirmation, etc.

Interruptability: Interruptable at any time. In fact it is often desirable to suspend scanning and structure a concept when a new and useful term is found, or where a knowledge base problem is detected.

Feedback potential: Low. The technique itself does not result in significant feedback, but feedback will normally result from interrupting it to structure concepts.

Group-efficacy: Suited to individual use, although the process of deciding how to deal with discrepancies can be effectively done with several experts involved.

Our experiences: We have had considerable success manually scanning documents to ensure that their terms and concepts correspond to those in the knowledge base; editing the document or knowledge base where necessary. So far we have used the computerised version of this technique only on small exapmples, however there has been considerable interest in it in connection with our ongoing BNR project.

Example: We took a software specification document and carefully scanned it. We found numerous subtle problems in *almost every* sentence. The process of correcting the knowledge base and the document was edifying to all concerned. Prior to this process, the document was considered 'correct' to the author and other experts, and the experts were slightly hostile to our attempts at critiquing. After the knowledge acquisition process (a large portion of which was simply discussion of the detected problems), the experts became very positive towards our technology, as they realized the hidden problems in their document.

- ***Conceptual diagram drawing***

  Starting point: Several existing types and properties (relations)..

  Result: An arbitrary graph of concepts: typically a mix of new concepts and refined concepts.

  Technique: The user manipulates a graph composed of nodes and links (see figure 3). The process normally starts with a few nodes (usually representing type concepts or instance concepts) on display. The user can add, delete and move nodes around by direct mouse manipulation; the user can also indicate conceptual relations by linking nodes together. Other useful features include the ability to rapidly alter the set of nodes and links that are displayed in order to work on different parts of the knowledge base.

  A number of knowledge base tools support this technique, e.g. KSSn (Shaw and Gaines 1991). In CODE4 we have added considerable sophistication for the experienced user, for example the graphing capability is tightly integrated with the outlining ability: The user can create networks of browsers such that concepts displayed in one window are functionally dependent on the user's selection of concepts in a prior 'driver' window. Updates in one window are automatically reflected in others, which may show the concepts in a different context.

  Interface modality: Almost completely non-modal.

  Interruptability: Completely interruptable.

  Feedback Potential: High. Typically the user will have several windows open on the knowledge base, and as changes are made in one window, the consequences can be seen in another window. CODE4 also provides a separate feedback window by which the system can comment textually on what has been entered.

  Group-efficacy: Generally an individual technique.

  Our experiences: We use diagram drawing extensively, although mostly for display with only minor editing. Most people find conceptual outline structuring (described below) to be more effective when a large number of changes have to be made.

  The most popular kind of diagram is the 'isa' hierarchy; second most popular are other kinds of hierarchical diagrams such as categorization of statements, and part decomposition. The ability to create other more complex diagrams is used far less, even though people are impressed when we demonstrate CODE2 and CODE4's capabilities..

  Example: When creating a model of a telephone system we used diagram drawing to show how parts are interconnected, and to draw a finite state diagram. All the knowledge contained in these diagrams could also be viewed by looking at properties of concepts using conceptual outline structuring (e.g. a state has in-transition and out-transition properties). This provided useful feedback.

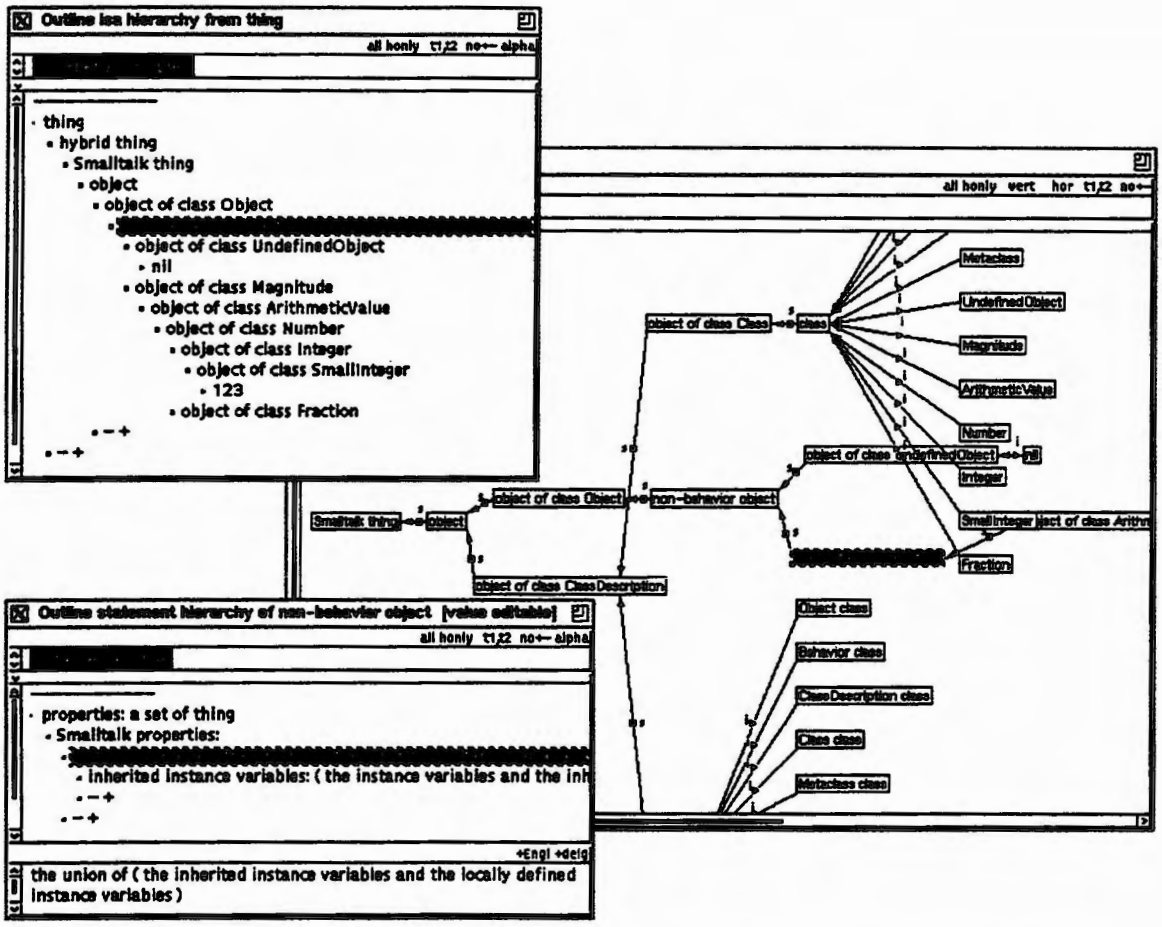


Figure 3: Two windows for conceptual outline structuring (left) and a window for conceptual diagram drawing (right). The knowledge is from our Smalltalk project and is expressed quite formally.

- ***Conceptual outline structuring***

  Starting point: Several existing types and properties.

  Result: As with conceptual diagram drawing, an arbitrary graph of concepts (although only a graph in the mathematical sense, not displayed as such)

  Technique: As with conceptual diagram drawing, the user uses direct manipulation to manipulate nodes and links. However the nodes are lines of text, and the relations between the nodes are indicated by indentation. This technique is largely a textual analog of conceptual diagram drawing, with an appearance somewhat like a conventional outline processor (see figure 3). In order to facilitate learning, the user inputs required in CODE4 to manipulate nodes and links have been made almost identical to those of the conceptual diagram drawing technique

  Interface modality: Almost completely non-modal.

  Interruptability: Completely interruptable.

  Feedback Potential: High, in the same way as conceptual diagram drawing.

  Group-efficacy: Generally an individual technique.

Our experiences: This technique is the most heavily used in the CODE2 and CODE4 repertoire. People use it to structure 'isa' hierarchies and the statement hierarchies that describe an individual concept.

The main differences between the diagraming and outlining techniques (aside from visual appearance) are the following:

1) Much more knowledge can be displayed in a window using outlining. This is both because of the reduction in white space, and because nodes can be physically larger, containing useful secondary information (this cannot be done in a diagram because it would prevent effective screen layout).

2) Impure hierarchies (e.g. those involving multiple inheritance) and arbitrary directed graphs are much more easy to manipulate using diagram drawing because it is two-dimensional

We find that people's preferred mode of thinking or SET (Meyer and Ross 1991) (visual or textual/aural) tends to influence choice of technique. This is why we claim that these techniques should be considered distinct, despite the fact that the users interactions are very similar: People definitely feel that they are using different techniques.

- ***Property table completion***

Starting point: A largely complete conceptual structure.

Property Comparison Matrix

Supers: OPTICAL DISC	VIDEODISC	compact disc
available recording surfaces	generally two, but sometimes only one.	n/a
content	video	one or more of: data, audio, graphics, still
dimensions/diameter	3.5, 5.25, 8, 12, or 14 inches	4.72 inches (12 cm)
error correction	no	no or yes: if yes; description of the error
laser type	gas laser or semiconducter laser	n/a
method of reading	the laser is either reflected off the	n/a
rotation technique	one of: CAV, CLV	CLV method
standard	one of: National Television System	one of Red Book, Yellow Book, Green
storage capacity	800 - 1000 MB	

Figure 4: A window for property table completion from the Cogniterm knowledge base. Note that most of the knowledge is very informal natural language.

Result: A refinement of the conceptual structure to fill in gaps in the knowledge.

Technique: Concepts are are listed on one axis, and properties are displayed on the other axis of a table (see figure 4). The values of statements (whose subject is a concept and whose predicate is a property) are displayed in cells of the table. The user can edit the table, particularly the values of the cells, somewhat like a spreadsheet.

Interface modality: Largely non-modal.

Interruptability: Completely interruptable.

Feedback Potential: Very high. Editing affects other windows and the feedback panel in the same way as with diagraming and outlining. This technique gives valuable feedback allowing the comparison of concepts, highlighting discrepancies. Numerous different tables can be dynamically generated by selecting appropriate sets of concepts and properties.

Group-efficacy: Generally an individual technique.

Our experiences: This has been a relatively new addition to our CODE2 repertoire and has been largely used to display tables with little editing. Users have found it prevents 'flipping' between concepts or opening large numbers of windows, and makes the process of comparing concepts much easier. Further research is needed, particularly the addition of the technique to CODE4.

- ***Systematic statement critiquing***

Starting point: A moderately complete knowledge base.

Result: A more complete knowledge base with 'problems' removed.

Technique: Statements in the knowledge base are presented to the participants for critiquing in a structured order. They are normally syntactically embellished to look like natural language. Statements may be immediately changed if problems are found, or the problems may be recorded so they can be fixed by individual experts. The choice of immediate or deferred updating substantially effects the sequencing of statements that should be presented.

We find it best to present a blocks of statements (e.g. a related subset of all those about a concept) on the screen and ask the participants to read them and speak up if they detect any problems. It is particularly important to be systematic in the order of presenting the statements, e.g. starting from the most general concepts (once the most general concepts are found to be correct, there are less likely to be problems with inherited knowledge).

We envisage expanding this technique when it is done in a group so it operates like a software code inspection; i.e. with paraphrasing and the discipline of pre-examining the knowledge base by participants.

When critiquing participants should ask the following questions of each statement: Is it: 1) Correct with respect to the opinion of the experts? 2) Consistent within the knowledge base? 3) Unambiguous in its expression? 4) Sufficiently formal to be useful? 5) Represented in the most appropriate way? 6) Sufficiently general or specific? 7) Sufficiently complete? and 8) Relevant to the purpose of the knowledge base?

Interface modality: Largely modal.

Interruptability: Interruptable, but it is best if interruption is not done excessively, as the idea is to be systematic in covering the knowledge base. If the user jumps out of se-

quence and edits knowledge elsewhere, it may be necessary to restart the process as previously validated knowledge may no longer be valid. Of course, correcting knowledge may have a similar effect: An adequately intelligent system would take this into account when sequencing the statements to be presented.

Feedback Potential: Moderate. The knowledge is presented, and any corrections may cause other displays to update, reflecting consequences. But as was noted under 'interruptability' it is best to focus on the current statement.

Group-efficacy: Typically performed individually; however 'inspection' of the knowledge base by several people in parallel can be useful for detecting and correcting problems that arise out of differences of opinion (i.e. consensus-building)

Our experiences: At the current time, we use the user interface of the outlining technique, and rely on the user's familiarity with the knowledge base to systematically present statements to the participants for critiquing. With future research we hope to have the system help organize the session.

Example: In the early phases of the BNR project, individual experts would enter knowledge using outlining and then get together for group critiquing sessions led by a knowledge engineer. CODE2 assisted by a) selecting material to be reviewed, and b) recording the status of group decisions about each statement. These sessions proved extremely valuable, although they were sometimes quite arduous as the knowledge was very complex.

- ***Extended inference***

Starting point: A largely complete knowledge base.

Result: Further refinement of the knowledge base.

Technique: This technique has much the same purpose as systematic critiquing: finding problems with the knowledge. The difference is that specialized automatic inference methods are used.

We assume that there are some basic inference methods that run in almost real-time, computing new knowledge (forward-chaining) or detecting problems (backward chaining). CODE4 has inheritance, delegation, value combination and a number of others. When a knowledge base nears completion, however, we may want to unleash the full power of first order logic to perform inferencing that is too computationally complex to perform in real time. We call this *extended inferencing*.

The user must request that extended inferencing be run. He or she may request that it be performed on a particular part of the knowledge base and may specify parameters such as search depth. The system then computes either forwards or backwards and reports its results. In the future we envisage the results being presented in such a way that the user can perform systematic critiquing of them.

Interface modality: Non-modal. The user is presented with a list of consequences with which to do whatever he or she wants.

Interruptability: This technique is inherently atomic from the user interface point of view, so interruption is not possible (of course, the on-going computation could be aborted, perhaps displaying results computed so far).

Feedback potential: Very high. Feedback is the *raison d'être* of this technique.

Group-efficacy: Not applicable. It only takes one user to start the inferencing (although subsequently a group may use the critiquing technique on the consequences).

Our experiences: We developed a Prolog-based adjunct to CODE2 called FOLDE (Full First-Order Logic Deduction Engine) (Skuce 1991). Interestingly, we found nobody but ourselves was interested in using it, and we have not yet used it in any of the major projects listed in section 1.2. We believe that this is partly due to the following: a) the need to fully formalize any input knowledge in order to use extended inferencing; b) the fact that its user interface is not yet adequately developed; and c) lack of familiarity with, and fear of, logic. Further research is needed.

Table 1 summarizes and quantifies some of the analysis of techniques given in this section. All of the attributes are rated on a scale of 0 to 10; the values assigned are the authors' judgements, and should primarily be considered relative to each other.

Technique	Average formality of starting point	Amount of new knowledge (vs. refined)	Average formality of result	Potential for creativity	Level of feedback provided	Group efficacy
Brainstorming	0	10	1	10	7	10
ISM	2	4	4	3	3	9
Scanning	1	8	6	1	1	3
Diagraming	5	6	8	9	8	4
Outlining	4	8	7	9	9	5
Table completion	7	3	8	5	10	5
Critiquing	6	1	9	3	5	6
Extended inference	10	2	10	2	10	6

Table 1: Ratings of some of the eight techniques of figure 1 in terms of the questions posed in section 2.1

Table 2 lists the knowledge engineering projects we described in section 1.2, and indicates the degree to which we used each technique. It also ranks some of the projects in terms of the questions asked in section 2.1. The columns have the following meanings:

X: Relative complexity of the knowledge bases produced, on a scale of 1-10

F: Formality of the resulting knowledge base, expressed as a percent.

N: Percent of time spent adding knowledge as opposed to refining or debugging it

C: Percent of the resultant knowledge base that was creative, (i.e. that contained knowledge that was not present anywhere in any form prior to the project)

G: Percent of work performed in groups

As with table 1, the figures in table 2 are the authors' judgements and should be considered in a relative sense. In future research we expect to instrument CODE4 so that some of these figures can be measured.

Project	X	F	N	C	G	Techniques Used	Comments
Telos	10	15	30	40	50	35% Outlining	In the early and late phases, experts did most of this. In the intermediate phase most was done by a knowledge engineer after reading documents and interviewing experts.
						55% Critiquing	In the early phases, this technique was performed intensively. During the time when the knowledge engineer was performing the outline browsing, the statement critiquing became easier.
						5% Diagraming	At the time, CODE lacked the diagram-drawing power it now has
						5% Scanning	Some design documents were scanned and the documentation group became interested in this technique to make sure that all terms were properly described in the knowledge base
Telebrain	6	30	70	90	5	65% Outlining	The knowledge engineer and expert were the same person, resulting in
						15% Diagraming	It became particularly useful to represent engineering diagrams directly in the knowledge base.
						20% Critiquing	This figure was lower than normal because this project was almost entirely inventive in nature, this there was little checking for correctness, only consistency, completeness, ambiguity etc.
Ontology	2	25	10	50	10	30% Outlining	
						10% Diagraming	
						60% Critiquing	The knowledge bases created for this project are typically very small, so the focus is on maximizing the correctness and utility of the statements
Self-description	4	15	45	60	30	5% Brainstorming	
						5% ISM	
						5% Scanning	During several design efforts, natural language documents were done first
						25% Outlining	
						10% Diagraming	
						30% Critiquing	
Cogniterm	7	10	65	10	20	30% Outlining	
						20% Diagraming	
						10% Table completion	This technique is particularly important when comparing concepts or writing definitions.
						30% Critiquing	
Smalltalk	4	35	60	20	5	70% Outlining	As with Telebrain, the knowledge engineer and expert were the same person
						30% Critiquing	

Table 2: How we applied the techniques to the projects described in section 1.2. See the text for a description of the lettered columns.

2.3 Other techniques

It is important to understand that we are not advocating the abandoning of many other tried and true techniques, such as those surveyed by Neale (Neale 1989). We have just not yet found it useful to integrate semi-automated support for them into our environment. We add techniques when there is sufficient demand from users, or when we have sufficient evidence that an experimental implementation would be worthwhile.

For example, there are many techniques in the literature to help a knowledge engineer conduct interviews. We try to encourage experts to work directly with our system by providing a simple and flexible user interface, and permitting very informal input. However, where interviews are necessary we see little advantage in computer assistance: A skilled interviewer can do an effective job of elicitation and paper-and-pencil recording of results. The computer integrated techniques come into play following interviews when the knowledge engineer typically uses outline structuring to enter the elicited knowledge.

Repertory grids are used in another popular technique (Gaines and Shaw 1987). We have, however, found only minimal use for them in the kind of knowledge engineering projects we do because of their limited expressiveness. The closest we come is our table completion technique, which is more general in one sense, but which does not provide the same kind of dynamic feedback. Interestingly, the knowledge gathered for the tables in this paper may have benefited from repertory grid analysis.

2.4 Combining the computer-supportable techniques into a methodology

Different knowledge engineering project require different techniques. As we pointed out in section 1.1, we focus on descriptive modeling projects, but even within this constraint, projects can require different techniques.

It is typical for 'methodologies' for various activities to prescribe a relatively fixed procedure to achieve some goal. We believe, however, that just about any technique can have its place and it is up to the skill of the knowledge engineer to pick the right technique and to switch back and forth between techniques as needs dictate. In our view, therefore, a good methodology should be no more than a set of heuristics that suggest when a given technique can be useful.

The most important heuristic we espouse is 'use the technique that appears the most useful at the time, and feel free to switch among techniques'. However, our analysis of the above techniques (including the analysis presented in the tables) has led us to develop a number of heuristics. The heuristics are still under development but the following issues are of major importance:

Role of knowledge engineer

- Skilled knowledge engineers tend to do most of their work using conceptual outline structuring. We encourage non-knowledge-engineers to directly use the system: They tend to also use outlining extensively, although with fewer of its powerful features. They also tend to want to see knowledge represented in forms such as diagrams and tables.

Sources of existing knowledge

- If the knowledge is largely in people's minds then brainstorming, ISM and conceptual structuring are effective at obtaining it. Quite often though, there is a substantial body of written material. We find it important to scan this written material, both to ensure its content is captured and to ensure that it is not inconsistent with the knowledge base. The latter problem is extremely common.

Complexity and amount of knowledge

- Knowledge bases can get very large and complex, and the classic 'knowledge acquisition bottleneck' can hinder further progress. With astute use of partitioning aids and feedback mechanisms in the conceptual structuring techniques it is possible to 'hide' much of the complexity. Systematic critiquing can prevent excessive iteration when trying to debug a complex knowledge base.

Phase of the project

- Typically the early phases of a project will involve more use of brainstorming and other techniques that are best at generating new knowledge. In later phases, the refining techniques are used more. We find it useful, though, to expand and refine the knowledge base iteratively, focusing on somewhat different areas each time. This idea has been explored by Shaw and Gaines (Shaw and Gaines 1991)

Intended use of the knowledge

- If the knowledge acquisition is being largely performed to help understand a domain, rather than for the development of an inference-performing system (e.g. the typical expert system), then less formality is needed in the resulting knowledge. This does not, however, mean that critiquing is not necessary; it means that the system need not 'understand' the representation of every concept.

3. INTEGRATION

We have used the following principles when integrating the techniques into CODE4:

1. All of the techniques operate in windows and act on an underlying knowledge base using a well-defined series of internal editing commands (i.e. there is a common API between all the user interface windows and the knowledge engine).

2. We have defined a higher-level API for techniques that involve displaying or traversing sets of related concepts (diagram drawing, outline processing, property table completion and systematic statement critiquing). This API has its own set of common internal commands operating on what we call 'knowledge maps' that each describe a particular view of a knowledge base.

3. The user interfaces for all the techniques have a simiiar look and feel.

4. Changes made in one window are, by default, reflected in others immediately

5. Windows for different techniques can be made to 'drive' one another with selections made in one window determining what is displayed in another window on a different technique.

4. SUMMARY AND CONCLUSIONS

In this paper, we have presented a series of knowledge acquisition techniques for which we are developing integrated computer support within CODE4. Some of these (e.g. Interpretive Structural Modelling, conceptual outline structuring) have not been extensively discussed in the knowledge acquisition literature and we therefore desire to point out their advantages.

The techniques vary in a number of dimensions. Some, such as brainstorming, and diagram- or outline-oriented conceptual structuring stimulate creativity. Some, such as table completion, extended inference and conceptual structuring give the users a large degree of feedback, helping them to understand the consequences of the entered knowledge.

Brainstorming and ISM are the techniques most suited to use in a group, although we have had considerable success with group-oriented critiquing. Brainstorming, conceptual structuring and scanning are most effective at generating large quantities of new knowledge; while critiquing, extended inference and table completion are the most useful for refining knowledge.

Conceptual outline structuring and critiquing are the techniques that are used most often. Technique usage depends strongly on factors such as the type and complexity of the project and the knowledge engineering skill and personal preferences of the users. We find that a large, complex project tends to involve much more use of the refinement-oriented techniques. A skilled knowledge engineer uses mostly conceptual outline structuring, although conceptual diagram drawing is used by visually-oriented thinkers and in cases where non-hierarchical relationships are important.

ACKNOWLEDGEMENTS

Support for this research has been provided by Bell-Northern Research and the Natural Sciences and Engineering Research Council of Canada.We would like to thank the following people for their work on various knowledge engineering projects: Ingrid Meyer, Karen Eck and Lynn Bowker for Cogniterm; Yves Beauvillé and Joe Sarkic for the self-description project; Ken Iisaka for the Smalltalk project, and numerous people at BNR for the Telos project, especially Roy MacLean and Milan Javor.

REFERENCES

Bradshaw, J., C. Chapman, et al. (1992). "An Application of DDucks to Bone-Marrow Transplant Patient Support". *1992 AAAI Spring Symposium on Artificial Intelligence in Medicine*, Stanford University, March.

Bradshaw, J., P. Holm, et al. (1992). "eQuality: A Knowledge Acquisition Tool for Process Management". *FLAIRS 92*, Fort Lauderdale, Florida.

Delbecq, A. L. and e. al (1975). *Group Techniques for Program Planning*. Glenview, IL, Scott, Foresman & Co.

Gaines, B. and M. Shaw (1987) *Knowledge Support Systems*, MCC.

Iisaka, K. (1992) *Knowledge Base of Fundamental Concepts of the Smalltalk-80 Language and Programming Environment*, Undergraduate Senior Report, University of Ottawa.

Lenat, D. and R. Guha (1990). *Building Large Knowledge Based Systems*. Reading, MA, Addison Wesley.

Lethbridge, T. C. (1991a). "Creative Knowledge Acquisition: An Analysis". *6th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff.

Lethbridge, T. C. (1991b). "A model for informality in knowledge representation and acquisition.". *Workshop on Informal Computing*, Santa Cruz, Incremental Systems.

Lethbridge, T. C. and D. Skuce (1992). "Informality in Knowledge Exchange". *AAAI Workshop on Knowledge Representation Aspects of Knowledge Acquisition*, San Jose.

Meyer, I., D. Skuce, et al. (1992). "Towards a New Generation of Terminological Resources: An Experiment in Building a Terminological KB". *13th International Conference on Computational Linguistics (COLING).*, Nantes.

Meyer, M. and J. Ross (1991). "Psychotherapeutic techniques for identifying experts' and system users' sensorial experience of thinking: Significance to knowledge acqusition". *proc. 6th Banff Knowledge Acqusiition for Knowledge-Based Systems Workshop*, Banff.

Miller, G. (1990). "WordNet: an On-line Lexical Database." *International Journal of Lexicography*. 3(4): whole issue.

Neale, I. M. (1989). "First generation expert systems: a review of knowledge acquisition methodologies." *The Knowledge Engineering Review* : 105-145.

Saunders, C. (1990) *A PRISM Handbook*, , BNR, Internal Document.

Shaw, M. and B. Gaines (1991). "Using Knowledge Acquisition Tools to Support Creative Processes". *proc 6th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff.

Skuce, D. (1991). "Interactive Debugging of First-order Knowledge Bases." *Data and Knowledge Engineering* submitted.

Skuce, D. (1992a). "Knowledge Management in Software Engineering: A Tool and and Experiment". *7th Knowledge Based Software Engineering Conf.*, Tyson's Corner, VA.

Skuce, D. (1992b). "Managing Software Design Knowledge: A Tool and an Experiment." *Resubmitted with reviewers changes to: IEEE Transactions on Knowledge and Data Engineering* : 36.

Skuce, D. (1992c). "A Wide Spectrum Knowledge Management System." *To appear in: Knowledge Acquisition* : 49 pp.

Skuce, D. and T. C. Lethbridge (1992). "A Knowledge Representation for Interactive Knowledge Management". *Submitted to: 3rd International Conf. on the Principles of Knowledge Rep. and Reasoning*, Cambridge, Mass., October.

Skuce, D. and I. Monarch (1990). "Ontological Issues in Knowledge Base Design: Some Problems and Suggestions.". *Proc. 5th Knowledge Acquisition for Knowledge Based Systems Workshop*, Banff.

Szpakowicz, S. (1990). "Semi-automatic acquistion of conceptual structure from technical texts." *Int. J. Man-Machine Studies* 33 385-397.

Warfield, J. N. (1976). *Societal Systems: Planning, Policy and Complexity*. New York, Wiley.

Using the Operational Modeling Language OMOS to Represent KADS Conceptual Models

Marc Linster
AI Research Division, GMD
PO Box 1316, D-5205 St. Augustin 1
marc.linster@gmd.de

Abstract

We present a modeling approach that integrates advantages of KADS-like conceptual modeling with advantages of knowledge acquisition approaches focussing on the development of directly operational systems. We describe a set of requirements for such an approach and present the operational modeling language developed to support it. We use the cancer chemotherapy administration task of ONCOCIN to illustrate our work. Finally, we discuss the advantages of using an operational modeling language such as OMOS in combination with a conceptual modeling language such as KADS.

1 Introduction

We present and discuss a modeling approach that aims at combining the advantages of KADS-oriented [Wielinga *et al.*, 1992] conceptual modeling—*modeling to make sense*—with formal modeling, that is, *modeling to build systems* as it is implemented for example in PROTÉGÉ [Musen, 1989]. To illustrate the ideas and concepts of the approach, and the modeling language that supports it, we use ONCOCIN's task of cancer-chemotherapy administration.

Section 2 develops the requirements that arise from the combination of conceptual modeling with formal modeling. Section 3 describes the operational KADS-oriented modeling language OMOS. It explains the core concepts of OMOS using a subtask of the task of ONCOCIN. Section 4 relates the KADS modeling terms to the terms of OMOS. Section 5 describes how a KADS conceptual model is transformed into an OMOS operational model. To illustrate this transformation process we use K-ONCOCIN, the KADS conceptual model of the cancer-chemotherapy administration task of ONCOCIN [Linster and Musen, 1991; Linster and Musen, 1992]. This large real-world task highlights the advantages of using an operational modeling language with certain formal properties, once one has developed a conceptual. Section 6 discusses the pros and cons of using OMOS for KADS.

2 Assumptions and Requirements for Operational Modeling Languages in a Knowledge Acquisition Context

Early work in knowledge acquisition for knowledge-based systems, for example work on ETS [Boose, 1985] or KRITON [Diederich *et al.*, 1987; Linster, 1989] emphasized a transfer view of knowledge acquisition. Knowledge was considered to be in an expert's head, and the knowledge engineer, using a tool, elicits the knowledge and represents it in a suitable operational formalism.

Today, we see knowledge acquisition as a constructive—even creative—modeling process (cf. [Wrobel, 1988; Morik, 1989; Clancey, 1989; Linster, 1992a] for arguments for and discussions of this point of view on knowledge acquisition). The interaction between domain expert, knowledge engineer, and tool creates knowledge. An explicit model represents a structured understanding of the entities and processes that contribute to the solution of a real-world task. The process of building such a description involves on the one side modeling to make sense of observed behavior and on the other side modeling to implement systems to reconstruct comparable problem-solving actions. The implementation of the behavior cannot be separated from the model construction process, as models tend to become so complex that their behavior can hardly be predicted on the basis of a non-executable document.

On the basis of personal experience in the construction of knowledge-based systems we define several requirements that we assume knowledge engineering environments must meet. [Linster, 1992c; Linster, 1992a] discuss them in more detail, and show how we derived OMOS from these requirements.

1. **Epistemologic modeling constructs.** The modeling language must provide a vocabulary that supports the knowledge engineer in the bottom-up process of conferring meaning to new concepts and situations by allowing him to construe such situations in the terms of a modeling vocabulary, consisting of epistemologic knowledge structuring constructs in the sense of Brachman [1979].

2. **Reusable templates.** We need template-like structures that the knowledge engineer can use when interpreting different situations that share structural aspects. For example, systematic diagnosis is a problem-solving method that has been identified in many domains. The reusable templates must be selectable and configurable. Arguments for the developments of such templates have been given by [Chandrasekaran, 1988]—calling them *generic tasks*—and by [Wielinga *et al.*, 1992]—calling them *interpretation models.*

3. **Multifaceted modeling.** To promote insight, response, discussion, and to provide a focus of attention, model building should distinguish between naturally occurring facets of a real-world task, such as, modality, task sharing, procedural aspects, static structures. Besides merely distinguishing between these facets, a modeling approach must promote the exploitation of the interaction between these facets. Linster and Musen [1991; 1992] show that the interaction between different points of view facilitates the creation of a new model by providing insights into complex behavior.

4. **Formal properties.** Automated knowledge acquisition tools need guidance, which is derived from the model descriptions. This implies that the models must have formal properties from which we can derive additional information, such as the knowledge needs of the method.

5. **Operational models.** Models of knowledge-based systems are too complex to anticipate their behavior on the basis of non-operational—pictorial or textual—descriptions.

Non-operational systems exclude the feedback that a running system provides from the ongoing model elaboration.

3 The Operational Modeling Language OMOS

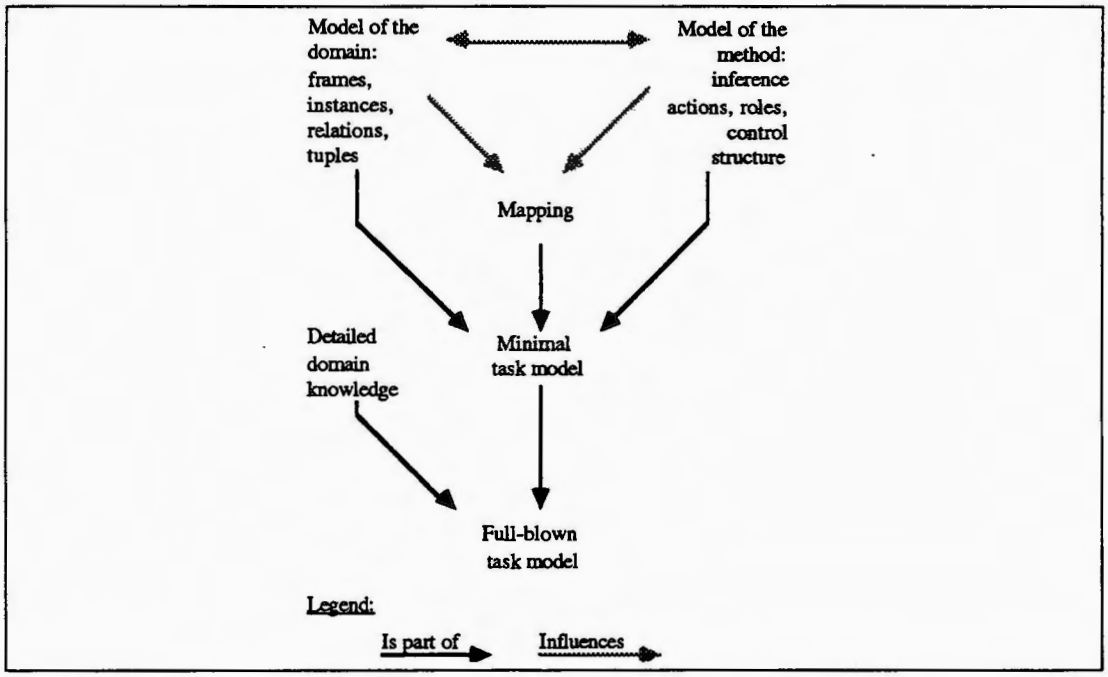


Figure 1: The interaction of the model of the method, the model of the domain, and the model of the task in OMOS.

We developed the operational modeling language OMOS to assess whether the requirements outlined in Section 2 can be fulfilled, and whether they lead to functioning and useful knowledge engineering environments.
In the model-building process, OMOS allows the description of observed phenomena from a problem-solving–method oriented point of view and from a domain-structure oriented point of view. The first one is described in the *model of the method*, the second one in the *model of the domain*. Other points of view, such as task-sharing are not considered in OMOS. Both models combine into the *model of the task* (see Figure 1), which is directly operational[1]. The model of the task can be executed, tested, and analyzed to provide feedback for the actual model-building process.

3.1 Laboratory Test Scheduling: A Sub-Task of ONCOCIN

To introduce the language elements of OMOS we use a small part of the cancer-chemotherapy administration task of ONCOCIN. The modeling of the whole task will be discussed in Section 5. ONCOCIN is a knowledge-based system that monitors the treatment of cancer patients assigned to standard treatment protocols. At recurring intervals it proposes a set of laboratory tests that should be effectuated on the current day of the currently administered

[1]We use the term *model of the task* in Musen's sense [Musen, 1989], meaning a method combined with domain terminology.

Table 1: The definition of the domain classes *therapy*, *chemotherapy*, *CcOM*, and *VAM*. *VAM.A.1* is an instance of *VAM*. It inherits from *VAM*, *chemotherapy*, and *therapy*.

```
(DEFFRAME Therapy
  (SUPERS DOMAIN-CONCEPT)
  (Slots (Days-In-Cycle 0
              :POSSIBLE-VALUES :NUMBER)
         (Day-Initiated 0
              :POSSIBLE-VALUES :NUMBER)

(DEFFRAME CcOM
  (SUPERS Chemotherapy)
  (SLOTS  (days-in-cycle 42)
          (drugs ((CCNU  vincristine
                         Methotrexate)))
          (delay-nxt-visit 7)
          (max-nbr-delays 2)))

(DEFINSTANCE VAM.A.1 of VAM)
```

```
(DEFFRAME Chemotherapy
  (SUPERS Therapy)
  (SLOTS (Drugs - :POSSIBLE-VALUES :LIST)))

(DEFFRAME VAM
  (SUPERS Chemotherapy)
  (SLOTS (days-in-cycle 21)
         (drugs ((VP-16 Adriamycin
                        Methotrexate)))
         (delay-nxt-visit 7)
         (max-nbr-delays 2)))

(DEFINSTANCE VAM.A.2 of VAM)
```

therapy. It uses the results of these tests to assess the patient's reactions to the treatment, and to derive dosage recommendations for the next visit. For the introductory example, we model reasoning steps such as: *If the chemotherapy VAM is currently being applied, and if we are in the first day of the cycle of application of that chemotherapy, then the laboratory test* wbc *(white blood cell count) is due.*
Section 5 contextualizes this reasoning step in the general task of ONCOCIN. Section 3.2 uses this subtask to introduce OMOS' modeling constructs for the domain; Section 3.3 presents the method-specific procedural concepts.

3.2 The Model of the Domain

The model of the domain is a structure-oriented view on the phenomenon to be analyzed. It looks for an infrastructure of types, their entities, and the relations among the entities. For the introductory example, we represent chemotherapies, laboratory tests, and knowledge about which test results must be available on which day of which chemotherapy.

Frames and Instances. To define the structures of the domain, we use an object-oriented approach, consisting of hierarchically defined classes—called *frames* in OMOS, with attributes—called *slots*, an inheritance relation, and instances of frames. In OMOS, we assume that classes—represented by frames—are used to provide structural definitions only. They are not modified in the reasoning process. Only instances are handled during problem-solving.
Table 1 renders the OMOS definition of several hierarchically organized frames and instances that are relevant for the task of scheduling laboratory tests. OMOS uses large parts of the frame language of the knowledge representation environment BABYLON V2.2 [Christaller *et al.*, 1992] to describe the structures and entities of a domain model.

Table 2: The definition of the relation `test-due`. `Test-due` declares which laboratory tests should be effectuated on which day of which therapy. The first statement of the definition says it is a binary relation. The second line declares the types of the arguments, that is, the first argument of every tupel is an instance of the frame *test*, such as *wbc* (white blood cell count), the second argument of the tupels is an instance-attribute-value triplet referring to instances of the frame *therapy* (see Table 1). The third line indicates prettier labels for the arguments. They are used in editors and analysis tools. The last argument lists the tupels of the relation.

```
(DEFINE-DOMAIN-RELATION Test-Due
  WITH ARITY = 2
       TYPE-SEQUENCE = (((INSTANCE Test)
                         (EXTENSION Therapy)))
       ARGUMENT-SEQUENCE = ((Laboratory-Test Day-in-the-Therapy))
       TUPLES = ((((Test-Due Wbc (_Vam-Occ Day-In-Cycle = 1))
                   <- (Vam _Vam-Occ))
                  ...
                  ((Test-Due Platelet (_Vam-Occ Day-In-Cycle = 1))
                   <- (Vam _Vam-Occ))
                  ((Test-Due Platelet (_Pocc-Occ Day-In-Cycle = 1))
                   <- (Pocc _Pocc-Occ))
                  ((Test-Due Platelet (_Pocc-Occ Day-In-Cycle = 8))
                   <- (Pocc _Pocc-Occ))
                  ...
```

Table 3: The list of extensional tuples that correspond to the first intensional tuple, given the instances of the frame *VAM* as rendered in Table 1.

```
(Test-Due Wbc (VAM.A.1 Day-In-Cycle = 1))
(Test-Due Wbc (VAM.A.2 Day-In-Cycle = 1))
```

Relations and Tuples. To express relationships among instances of different classes, OMOS provides relations. Relations consist of a declaration and of tuples. The declaration part describes the data-types of the arguments of the tuples. Tuples link instances and instance-attribute-value triplets. We distinguish between intensional descriptions of tuples—characterized through Prolog predicates—and extensional definitions through explicit listings of the tuples. The relation `test-due` of Table 2 is defined intensionally, i.e., through a list of predicates with variables. These predicates are true for those tuples that belong to the relation. The first predicate of the relation `test-due` uses a variable called `_vam-occ`, referring to all known instances of the chemotherapy *VAM*. If one uses the instances of the frame *VAM* defined in Table 1, the intensional definition corresponds to the list of extensional tuples rendered in Table 3.

3.3 The Model of the Problem-Solving Method

The model of the problem-solving method represents a process view on the real-world phenomenon, that is, a view documenting changes and (intermediary) states. To be able to compare methods of problem solving, to reuse them, and maybe develop a catalog of methods, the methods must be phrased independently from the domain that they are currently being applied to; they must be phrased in method-description terms that are characteristic for the point of view that they represent. To support the constructive bottom-up process

of model-building, the models of methods are built from primitive elements. The primitives that OMOS uses for the description of the method are KADS [Wielinga *et al.*, 1992] oriented. A method consists of a set of INFERENCE ACTIONS[2]—comparable to KADS knowledge sources, a set of roles—quite like KADS metaclasses, and a *control structure*, which is somewhat comparable to KADS task structures. The abstract data dependency between roles and inference actions is expressed in the *inference structure* (see Figure 2).

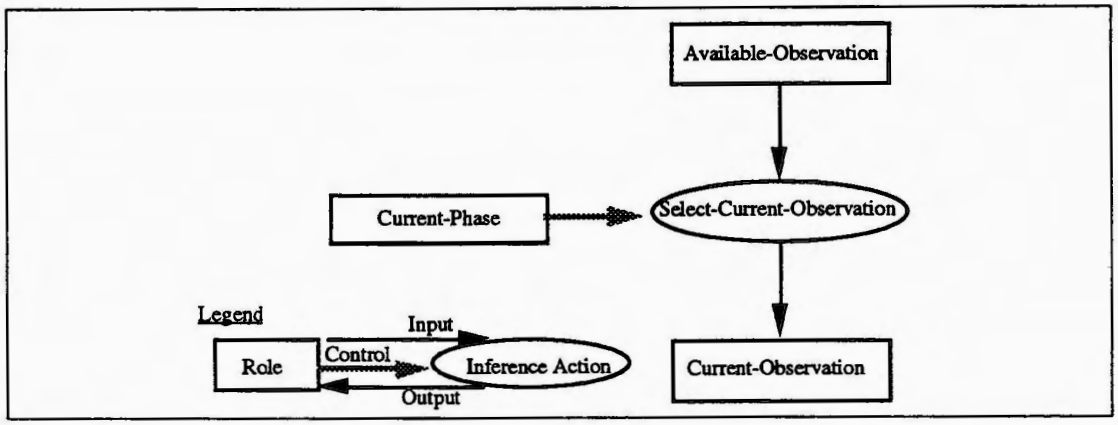


Figure 2: A small inference structure that shows the interaction between the inference action SELECT-CURRENT-OBSERVATION, and the roles available-observation, current-phase and current-observation. The elements of the role available observation are consumed, and are assigned the role current-observation, given that the contents of the control role current phase allows for it.

Roles. In a problem-solving process domain elements play varying roles. For our application example this means that a system knows about a set of laboratory tests, from which it selects certain tests on certain days. The fact that at first the system simply knows about a laboratory test and that later on this same test is assigned a specific part in the problem-solving process is expressed through a differentiation in roles. Initially, all laboratory tests known to the system play the role available-observation. Once they have been selected for effectuation on the current day they are denoted by the role current-observation.
Role assignments are implemented as dynamic predicates in OMOS. For example, initially the following holds: `(CURRENT-ROLE Available-Observation Wbc)`, stating that the laboratory test *wbc* (white blood cell count) plays the role of an available-observation; as soon as it is selected for effectuation it changes its role and becomes a current-observation.

Inference Actions. An inference action is a generic description of elementary change occurring during problem solving. An inference action operates on domain elements that are currently described by roles. Whereas KADS differentiates between *input* and *output* metaclasses for each knowledge source, OMOS further differentiates between *input* and *control* roles of an inference action. Input roles are read destructively, that is, elements that played this role, and that are involved in the firing of the inference action, are assigned to the output role. Control roles however, are read only; they are not modified by the inference action.
For the inference structure of Figure 2 this means that the role available-observation is accessed destructively as it is the input role for the inference action SELECT-CURRENT-OBSERVATION. Its domain instances are transferred to the role current-observation if the

[2]To ease readability, we use special type-setting conventions for INFERENCE ACTIONS, roles, *domain concepts*, and `domain relations`.

Table 4: The definition of the inference action SELECT-CURRENT-OBSERVATION (see also the inference structure in Figure 2). Lines 1–3 define the part that the surrounding roles play. Line 4 states that the inference action moves domain elements from its input role to its output role (if it fires). Line 5 says that it does not assign values to attributes of domain instances.

```
(DEFINE-INFERENCE-ACTION Select-Current-Observation
  WITH
(1)       INPUT-ROLE = Available-Observation
(2)       Output-Role = Current-Observation
(3)       Control-Roles = ((Current-Phase))
(4)       ROLE-ASSIGNMENT = TRANSFER
(5)       VALUE-ASSIGNMENT = FALSE
```

content of the control role current-phase allows for it. Table 4 renders the definition of this inference action.

In the context of OMOS, description of change is limited to (1) value assignments and (2) role assignments. In the case of a value assignment, the value of an attribute of a domain instance is modified. If an inference action implements a role assignment, then it assigns a new role to a domain instance by transferring it from its input to its output role.

An inference action operates on domain instances that are currently described by its input roles and it uses a domain relation. The specification of the inference action describes the nature of the change to be inflicted in terms of value or role assignments. The domain relation contains the detailed information about instances, attributes, and values needed to implement the change. The description of the actions in terms of changes in role or value is generic. If the generic description is combined with domain knowledge, it becomes task specific.

Control Statements. In OMOS, we view control as being part of the method definition. Thus, it must be described generically in terms of inference actions and roles. A control statement can (1) call an inference action, that is, activate the inference action and try to fire the forward chaining rules that implement it; (2) test whether an inference action inflicted any change, that is, test whether any of the underlying forward chaining rules fired; or (3) test whether a role currently denotes any domain instance.

Even though this control language may seem very limited, it has proven adequate for the modeling of a number of problem-solving methods, such as skeletal-plan refinement (see Section 5), selection by elimination and optimization [Kühn *et al.*, 1991], and straightforward assignment [Linster, 1992d].

3.4 The Model of the Task

The model of the task results from the combination of the model of the method with the model of the domain (see Figure 1). It is not defined explicitly; it results from the mapping of the method elements onto the elements of the domain model. Inference actions are mapped onto domain relations; roles are mapped onto domain concepts. Figure 3 shows how the elements of the domain model (e.g., *wbc*, *Karnofsky status*) are linked to the model of the method (e.g., *available-observation*). For the definition of the inference action SELECT CURRENT OBSERVATION (see Table 5) this means that Lines 1 through 5 are the generic, solely method-specific parts, whereas Lines 6 through 10 link the method definition to the domain, and thus are task-specific. Inference actions are automatically compiled into forward

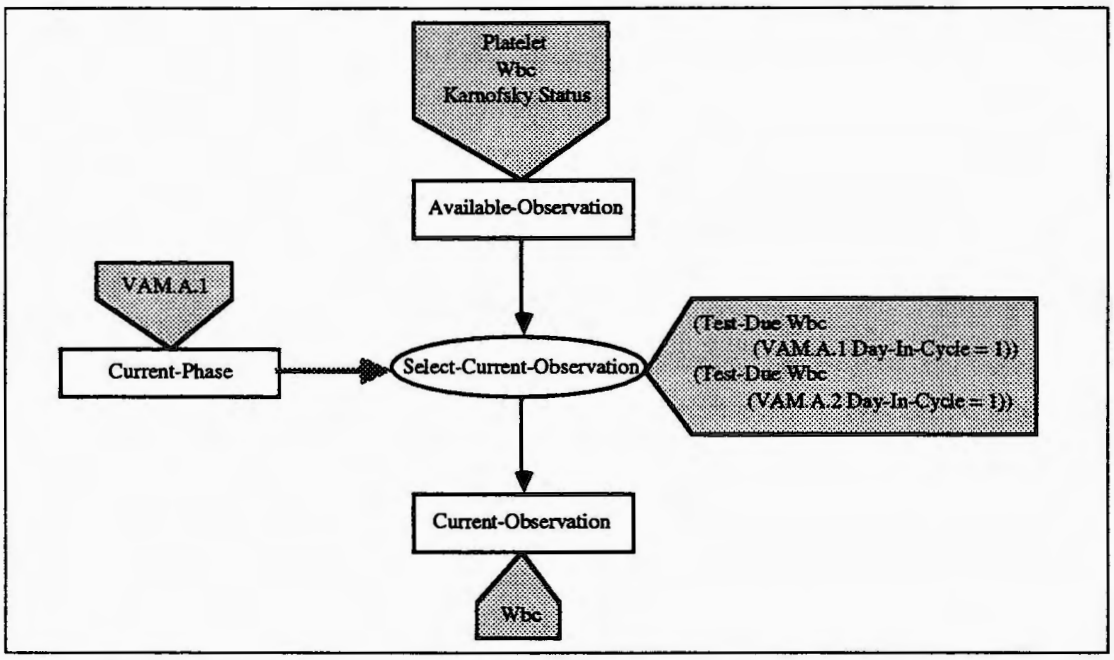


Figure 3: The inference structure of Figure 2 with some of the domain knowledge that it uses at some point in the problem solving process.

Table 5: The definition of the inference action SELECT-CURRENT-OBSERVATION with its task-model specific attachments to the domain model. (See Table 4 for the definition of the generic parts.) Line 6 states that it gets its detailed knowledge about which instances to handle and how to execute the changes of Lines 5 and 6 from the domain relation **test-due** (see also Tables 2 and 3). Line 7 reiterates the argument structure of the relation **test-due**. Lines 8, 9, and 10 describe the projection of the roles onto the arguments of the domain relation. For example Line 8 says that the input role is projected onto the zeroeth argument of the relation **test-due** (i.e., onto an instance of the class *test*).

```
(DEFINE-INFERENCE-ACTION Select-Current-Observation
  WITH
(1)       INPUT-ROLE = Available-Observation
(2)       Output-Role = Current-Observation
(3)       Control-Roles = ((Current-Phase))
(4)       ROLE-ASSIGNMENT = TRANSFER
(5)       VALUE-ASSIGNMENT = FALSE
(6)       DOMAIN-RELATION = Test-Due
(7)       REL-ARG-TYPE = (((0 . INSTANCE)(1 . EXTENSION)))
(8)       PROJ-ROLES-REL-ARGS = (((INPUT-ROLE  . 0)
(9)                               ((CONTROL-ROLES . 0) . 1)
(10)                              (OUTPUT-ROLE . 0))))
```

Table 6: The forward chaining rule schema of the inference action SELECT-CURRENT-OBSERVATION, instantiated with the detailed knowledge of the first tuple of the domain relation **test-due** (see Tables 2 and 3).

```
IF   (CURRENT-ROLE Available-Observation Wbc)
     (CURRENT-ROLE Current-Phase VAM.A.1)
     (VAM.A.1 Day-In-Cycle = 1)
THEN (TRANSFER-ROLE Available-Observation Current-Observation Wbc)
```

Table 7: A juxtaposition of the modeling terms that KADS and OMOS use.

KADS		OMOS	
Domain layer	Concept, relation	Domain model	Frame, instance, relation, tuple
Inference layer	Knowledge source, metaclass, inference structure	Method model	Inference action, role, inference structure
Task layer	Task, sub-task, task decomposition	Method model	Control structure
Strategy layer	Goals, plans, repair rules	*No equivalent*	*No equivalent*
Conceptual model		Task model	

chaining rule schemata that implement the changes described in the definition of the inference action, using the detailed knowledge described in the tuples of the inference action's domain relation. Table 6 renders the forward chaining rule schema of the inference action SELECT-CURRENT-OBSERVATION, instantiated with the first tuple of Table 3.

The model of the task is directly operational and it describes how domain layer knowledge is used in the framework of the problem-solving method. Section 5.4 describes how special tools use the interaction between the domain and method models to guide the knowledge acquisition process, once a minimal task model has been developed.

3.5 Limits of OMOS

As stated earlier, OMOS inference actions represent change as role assignment and as value assignment. Thus, all problem solving methods must be configured from these primitives. Even though a number of applications have been modeled this way, it is obvious that other primitives, such as instance creation, instance deletion, or structure modification will eventually be required to provide a broadly applicable modeling language. OMOS provides controlled exits so that single inference actions that do not fit OMOS' framework of change can be described through manually programmed bodies of code. This is comparable to the knowledge source code bodies of MODEL-K [Karbach *et al.*, 1991].

Furthermore, OMOS is limited to tasks that do not require the dynamic creation or deletion of domain instances. The limits and their consequences are discussed in detail in [Linster, 1992b; Linster, 1992c].

4 OMOS and KADS

Table 7 shows the juxtaposed terms of KADS and OMOS models. The structure of OMOS models is similar to KADS models. Both distinguish between an explicit description of the domain—called *domain model* in OMOS and *domain layer* in KADS—, an explicit description of the inferences and intermediate results or states—called *inference actions*, respectively *roles* in OMOS and *knowledge sources*, respectively *roles* in KADS—and a generic description of how these inferences are used to solve complex problems—called *control structure* in OMOS and *task structure* in KADS. We chose labels different from those of KADS to emphasize that the OMOS language elements are different in nature. KADS language elements are used to describe a conceptual model, which draws its semantics from the intention of the developer or the expectation of the reader. OMOS, however is a high-level programming language. OMOS language elements do not leave any room for conceptual interpretation. Their semantics are defined through the underlying implementation.

KADS talks about layers in the conceptual model (i.e., domain layer, inference layer, task layer, and strategy layer). OMOS talks about (partial) models (i.e., model of the domain, model of the method, and task model). They represent points of view onto a real-world task to be modeled which interact when a model is being developed. Similarly, the layers of a KADS conceptual model are not independent of each other, as for example the domain layer provides those distinctions that the inference layer relies upon, and vice versa, the inference layer indicates sets of distinctions that the domain layer must provide. The OMOS task model adopts Musen's definition [1989] of the term task (i.e., a method plus domain terminology). This may be confusing in a KADS context.

The KADS modeling approach emphasizes the modeling-to-make-sense aspects and the human communications aspects that are relevant in early knowledge acquisition phases, OMOS emphasizes the interaction between humans and computers. Humans act as creative agents that build the model; computers are rigorous agents that test the model as a whole, enforcing disambiguation and completeness. This explains why OMOS can be a suitable means to implement KADS conceptual models. The structure of the models in both languages is largely identical, but there is a shift of focus in the knowledge engineering process at some point, away from conceptual issues, towards implementational and formal problems. The next section shows how a large KADS conceptual model can be represented in OMOS' terms, and how this transformation from a conceptual model into a high-level operational model facilitates its elaboration and completion.

5 OMOS for KADS: Representing K-ONCOCIN in OMOS

This section describes how K-ONCOCIN—the KADS conceptual model of the cancer-chemotherapy administration task of ONCOCIN—is represented in OMOS' terms. The resulting OMOS model is then called O-ONCOCIN.

Section 5.1 gives a short overview of K-ONCOCIN. Due to limited space, we must refer the reader to [Linster and Musen, 1991; Linster and Musen, 1992] for a detailed description of the KADS model and its development process. Section 5.2 uses core aspects of K-ONCOCIN to illustrate the transformation of K-ONCOCIN into O-ONCOCIN. Section 5.3 summarizes the changes to the model that occurred when moving from KADS to OMOS. Section 5.4 uses O-ONCOCIN to illustrate the importance of the task model in OMOS. We describe how the task model is used to foster the ongoing model-building process and to show why it is important to re-focus the knowledge engineering process once a conceptual model has been developed: away from conceptual modeling, towards system building on the basis of high-level operational constructs.

5.1 K-ONCOCIN: The KADS Conceptual Model of the Cancer-Chemotherapy Administration Task of ONCOCIN

K-ONCOCIN is the KADS conceptual model of ONCOCIN: an automated mechanism to monitor the treatment of cancer patients assigned to standard treatment protocols. It includes a database of specifications for numerous protocols and a database of test and treatment records for numerous patients. The system assists the treating physician by providing recommendations for patient therapy and for the requisition of tests according to protocol guidelines. On the basis of (1) the latest laboratory-test results for a particular patient, (2) information from the most recent clinical examination, (3) the patient's treatment history, and (4) the specific treatment protocol to which the patient is assigned, ONCOCIN computes a recommendation for the next cycle of treatments and laboratory tests for the patient. Section 5.1.1 summarizes the domain layer knowledge of K-ONCOCIN; Sections 5.1.2 and 5.1.3 describe the inference and task layers of K-ONCOCIN, which implement a skeletal-plan refinement problem-solving method [Friedland and Iwasaki, 1985].

5.1.1 The Domain Layer of K-ONCOCIN

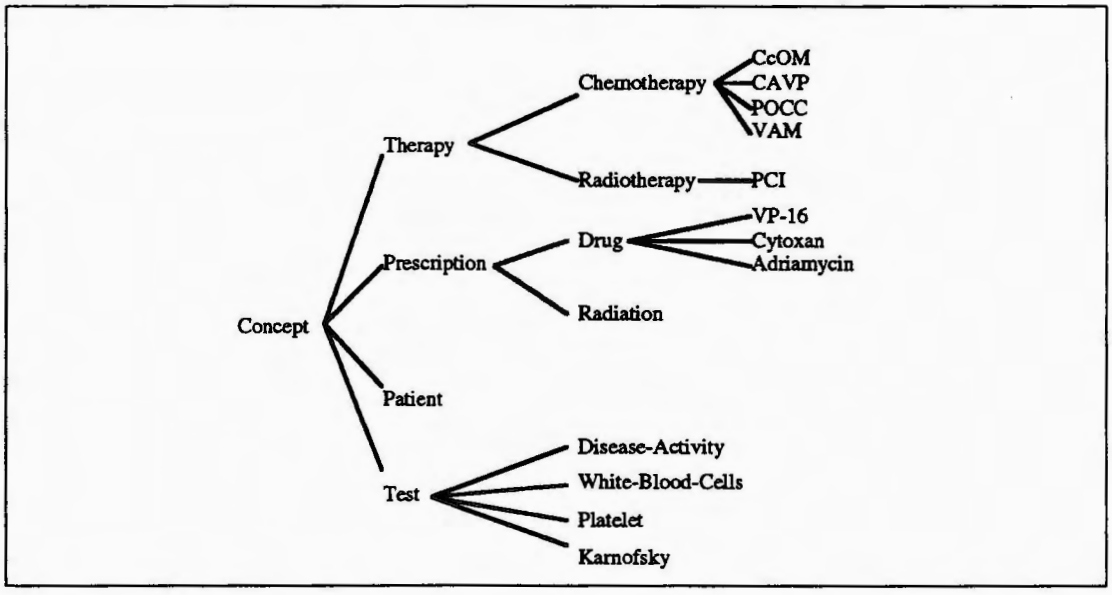


Figure 4: The hierarchical definition of the domain-layer concepts in K-ONCOCIN. We do not distinguish between concepts and instances of the concepts.

The domain layer of K-ONCOCIN consists of hierarchically organized concepts (see Figure 4), and of domain relations that represent interactions among concepts:
`Successor-Therapy` describes the sequence of possible therapies (i.e., chemotherapies and radiotherapies) on the basis of data about the patient (e.g., in Arm B of Protocol NCOG 2083-1 CAVP is followed by CcOM if the cancer is progressing).
`Drug-Due` indicates the days of particular therapies on which particular drugs are administrated (e.g., VP-16 is due on the first day of the chemotherapy VAM).
`Test-Due` indicates the days of a particular therapy on which a particular test is warranted (e.g., a WBC is due on the first day of the chemotherapy VAM).
`Drug-Dose-Attenuation` relates the results of laboratory tests to the dosage of drugs (e.g., in Protocol 2083-1, the dose of the drug VP-16 in the chemotherapy VAM is reduced to 75

percent of the normal dose if the WBC is below $3.5 \star 10^3$; if the WBC is above $3.5 \star 10^3$ and the platelet count is between 100 and $150 \star 10^3$, then the patient should receive 75 percent of the standard dose of Cytoxan in the chemotherapy CAVP).

`Chemotherapy-Alteration` relates the results of laboratory tests to the continuation or interruption of the current chemotherapies (e.g., if the patients platelet count is less than $100 \star 10^3$, then the current chemotherapy is delayed) The duration of the delay is a property of the chemotherapy, just as is the maximum number of delays permitted before the chemotherapy is aborted.

`Abstract-Disease-Activity` relates descriptions of how the patient responds to the treatment to more abstract findings (e.g., if the disease activity (a numerical representation of the state of the patients tumor over time) is equal to 6, then the patient is considered not to be responding to treatment, and the cancer is considered to be continuing to progress; if the value of disease-activity is equal to 1, then the patient is considered to show complete response to the treatment, and the cancer is considered not to be progressing).

5.1.2 The Inference Layer of K-ONCOCIN

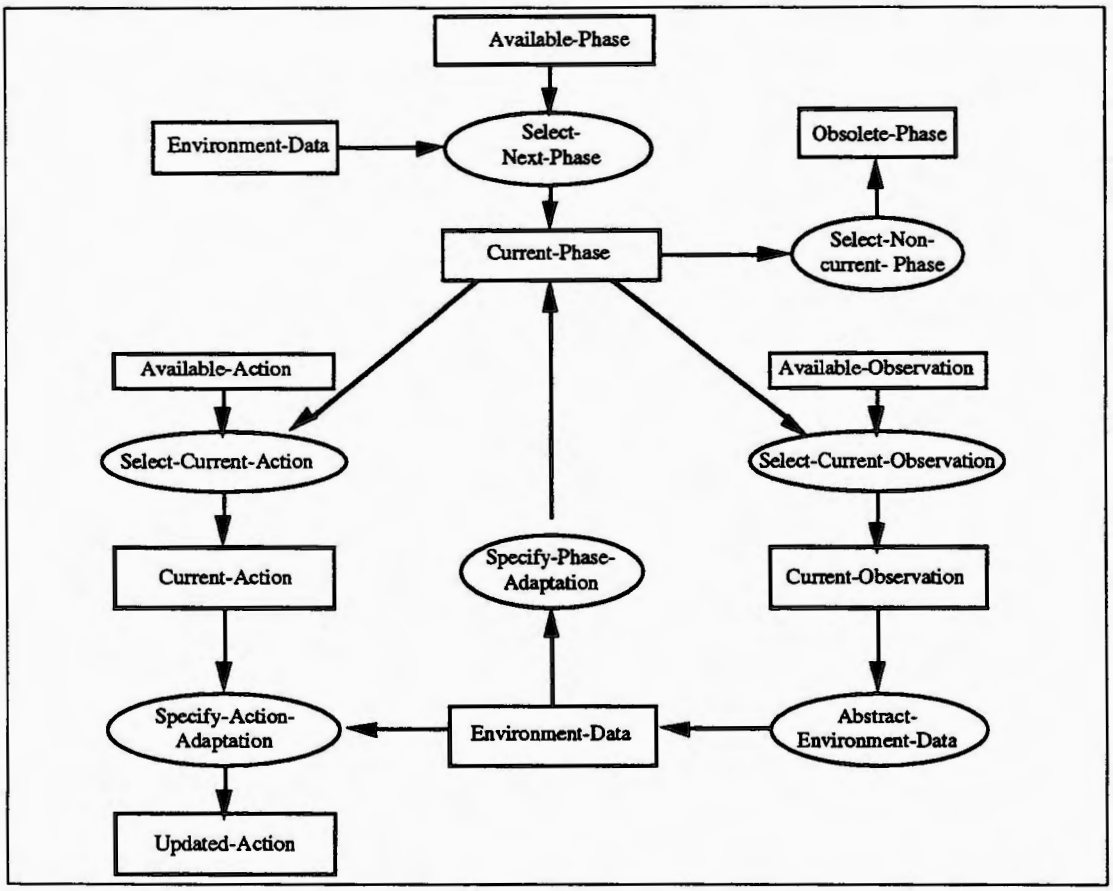


Figure 5: The inference structure of K-ONCOCIN (adapted from [Linster and Musen, 1992]).

In the task of ONCOCIN one observes five domain types that play varying roles in the problem solving process, thus defining metaclasses: the patient is described by the metaclass environment-data; drugs are denoted by the abstract label action; chemotherapies are

Table 8: The KADS task structure for K-ONCOCIN (adapted from [Linster and Musen, 1992]). Note that *Obtain (Current-Observation)* is an input task, *not* a knowledge source. This statement describes that the system triggers an external query to obtain data.

```
Task: Skeletal Plan Refinement
Goal: Refine a skeletal plan by selecting the next phase and evaluating it.
Task structure :
  WHILE SELECT-NEXT-PHASE (Available-Phase + Environment-Data ⟶ Current-Phase)
    DO
      WHILE NOT EMPTY Current-Phase
        DO
          IF SELECT-CURRENT-OBSERVATION
              (Available-Observation + Current-Phase ⟶ Current-Observation)
            THEN Obtain (Current-Observation)
                 ABSTRACT-ENVIRONMENT-DATA
                   (Current-Observation ⟶ Environment-Data)
                 SPECIFY-PHASE-ADAPTATION
                   (Environment-Data ⟶ Current-Phase)
          SELECT-NONCURRENT-PHASE
           (Current-Phase ⟶ Obsolete-Phase)
          IF SELECT-CURRENT-ACTION
              (Available-Action + Current-Phase ⟶ Current-Action)
            THEN SPECIFY-ACTION-ADAPTATION
                   (Environment-Data + Current-Action ⟶ Updated-Action)
```

described by the phase; laboratory tests are called *observations.*
The inference structure of K-ONCOCIN (see Figure 5) can be subdivided into several parts. The knowledge source SELECT-NEXT-PHASE determines the content of the metaclass current-phase, which denotes the currently administered chemotherapies. SELECT-CURRENT-OBSERVATION determines the content of the metaclass current-observation (i.e., the laboratory tests to be effectuated today) on the basis of all known laboratory tests and the current-phase, that is, the currently administered therapies. Similarly, SELECT-CURRENT-ACTION picks the drugs to be given today. They are denoted by the metaclass current-action. SPECIFY-ACTION-ADAPTATION uses knowledge about the patient's current status—denoted by environment-data—to attenuate the dosage of chosen drugs.

5.1.3 The Task Layer of K-ONCOCIN

Table 8 renders the task structure for K-ONCOCIN. It states that while the knowledge source SELECT-NEXT-PHASE is able to define a next phase (i.e., a therapy in our case), one selects observations (i.e., laboratory tests) and actions (i.e., drugs), and adapts the actions to the observations (in the knowledge source SPECIFY-ACTION-ADAPTATION).

5.2 Rephrasing K-ONCOCIN as O-ONCOCIN

This section describes how we transformed the KADS conceptual model into an executable OMOS model. As discussed in Section 4, OMOS models have the same structure as KADS conceptual models. This means that there will be a one-to-one translation from layer constructs on the KADS side to model constructs on the OMOS side. As K-ONCOCIN is a large model, we focus on some examples for every layer.

5.2.1 Tweaking the Domain Knowledge

KADS concepts become frames and instances; KADS relations are redefined in terms of argument types, arities, and tupels. OMOS requires a clear-cut distinction between the definition of structures—the frames—and the domain elements that can actually be handled in the process—the instances of the frames. This results in definitions as rendered in Table 1. The modeling with OMOS shows that it is not sufficient to model just the chemotherapies, but we must represent the different occurrences of the chemotherapies, as for example the third occurrence of the chemotherapy POCC is followed by PCI (prophilactic cranial irradiation), whereas the first and second occurrence of POCC are followed by the chemotherapy VAM. This sloppiness of the original conceptual model was uncovered when on the one side we had to provide unambiguous domain knowledge, and on the other side, when we could test the model. This leads to a richer definition of the domain knowledge (see Figure 6).

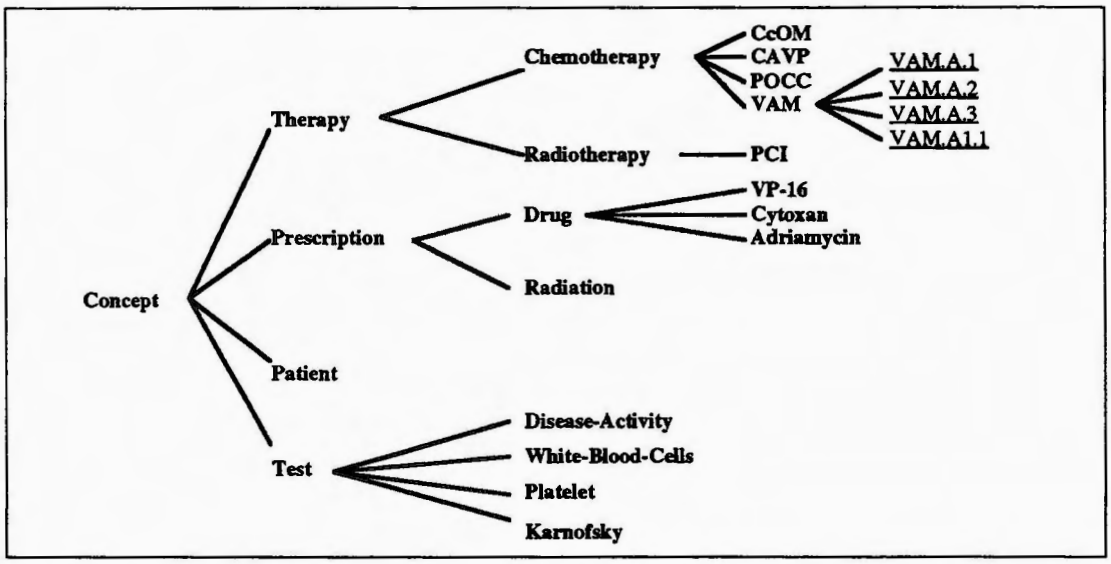


Figure 6: A part of the hierarchy of frames (set in bold) and instances (underlined) for the domain model of O-ONCOCIN.

5.2.2 Rewriting the KADS Inference Layer as a Part of an OMOS Method Model

Translating Knowledge Sources into Inference Actions We pick two representative knowledge sources and translate them into OMOS inference actions. The knowledge source SELECT-CURRENT-OBSERVATION modifies the roles that domain elements play in the problem-solving process; the knowledge source SPECIFY-ACTION-ADAPTATION is slightly more complex, as it modifies roles and attribute values of domain elements.

SELECT-CURRENT-OBSERVATION. In the KADS model, SELECT-CURRENT-OBSERVATION has two input meta classes: available-observation and current phase (see Figure 5). The elements of current-phase—the therapies—determine which available-observations—the laboratory tests—become current-observations. This means that in the OMOS representation, current-phase is used for control purposes. Available-observation and current-observation become input, respectively output roles. As the role assignment of domain elements is modified—they *flow* from available-observation to current-observation—one of the changes that the inference action performs can be characterized as ROLE-ASSIGNMENT = TRANSFER.

Table 9: The definition of the inference action SPECIFIY-ACTION-ADAPTATION. Lines 1 through 3 declare the roles. Line 4 relates the fact that domain elements (drugs in this application) change their role; Line 5 describes that the inference action assigns values to attributes of domain elements (in this case the attribute drug-dosage of selected drugs is assigned a value.)

```
DEFINE-INFERENCE-ACTION Specify-Action-Adaptation
  WITH
(1)       INPUT-ROLE = Current-Action
(2)       CONTROL-ROLES = ((Environment-Data))
(3)       OUTPUT-ROLE = Updated-Action
(4)       ROLE-ASSIGNMENT = TRANSFER
(5)       VALUE-ASSIGNMENT = INITIAL
```

Table 10: The definition of the role Available-Observation with its initial role assignment, stating that all instances of the class *test* play this role when the problem solving process starts out.

```
(DEFINE-ROLE Available-Observation
     WITH INITIAL-ROLE-ASSIGNMENT = ((Test _test-instance)))
```

The inference does not affect any attributes of the domain elements that are manipulated. Thus VALUE-ASSIGNMENT = FALSE is a correct characterization.
The complete definition of the OMOS equivalent of this knowledge source is given in Table 4.
SPECIFY-ACTION-ADAPTATION. SPECIFY-ACTION-ADAPTATION takes selected drugs—denoted by current-action and defines their current dosage, on the basis of data about the patient—contained in environment-data. This knowledge source describes inferences such as: *the dose of the drug VP-16 is reduced to 75 percent of the normal dose if the WBC is below* $3.5 \star 10^3$. Thus environment-data is a control role. As drugs change their role (i.e., they were denoted by current-action and through this inference become adapted-actions) ROLE-ASSIGNMENT = TRANSFER is an accurate characterization. Furthermore, the inference action modifies the dosage of the drugs to be administered. VALUE-ASSIGNMENT = INITIAL accounts for this.
Table 9 renders the definition of the inference action.

Rewriting the Metaclasses as Roles. In OMOS roles are implemented as dynamic predicates that are true for those domain instances that currently play the role. Some roles are initialized, for example available-observation denotes initially all laboratory tests (see Table 10), others are defined dynamically.

The OMOS Inference Structure for the ONCOCIN Task. Figure 7 renders the OMOS equivalent of the inference structure of K-ONCOCIN (see Figure 5). Several things have changed:

1. The OMOS inference structure makes an explicit difference between input, control, and output roles for every inference action.

2. The KADS task *Obtain (Current-Observation)* is modeled as an inference action.

3. The inference action INITIATE-CURRENT-PHASE has been introduced to initialize several properties of selected therapies. This is needed for computational reasons to reset the

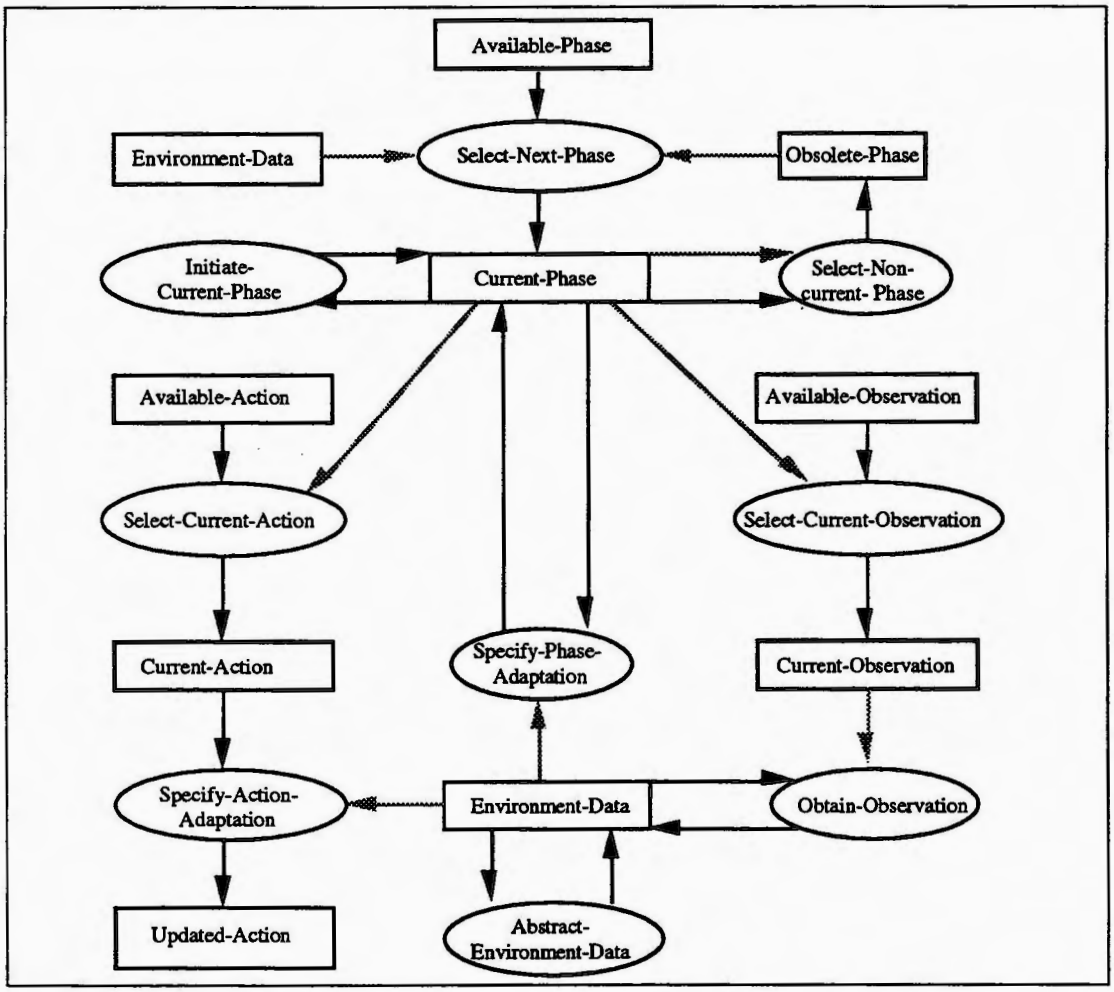


Figure 7: The OMOS inference structure for the task of ONCOCIN. The general organization is similar to the one of the KADS inference structure (see Figure 5), except for the additional distinctions between input, output, and control connections between roles and inference actions.

counters that are used to determine the current day in the cycle of the therapy.

4. SELECT-NEXT-PHASE uses obsolete-phase as control role. The inference action needs to know about past, that is, obsolete therapies to select the next therapies. This data flow is missing in the KADS model, but it should be there.

5.2.3 Representing KADS Task Structures as OMOS Control Structures

Table 8 renders the KADS task structure of K-ONCOCIN. Except for the input task *Obtain (Current-Observation)* it can be transformed into an equivalent OMOS control structure in a straightforward way (see Table 11). OMOS models the task *Obtain (Current-Observation)* as an inference action, called OBTAIN-OBSERVATION.
KADS task statements such as SELECT-CURRENT-OBSERVATION (Available-Observation + Current-Phase ⟶ Current-Observation) are rephrased as (`CALL-IA select-current--observation`), where `CALL-IA` stands for an activation of an inference action. The `WHILE` loops of the KADS structure are phrased in terms of `LOOP` and `RETURN` (or in any other

Table 11: The OMOS control structure for the model of the ONCOCIN task. Statements such as LOOP, COND, and RETURN are LISP control statements. The LOOP and RETURN statements in Lines 1, 5, 17, 18 operationalize the WHILE statements of the KADS task structure. CALL-IA is an activation of an inference action. ROLE-P tests whether a role still describes at least one domain element. The statements RESET-ROLE are new; they do not have equivalents in the KADS task structure.

```
(1)    (LOOP
(2)    (COND((Call-Ia Select-Next-Phase)
(3)          (CALL-IA Initiate-Current-Phase)
(4)          (CALL-IA Select-Noncurrent-Phase)
(5)          (LOOP (COND ((ROLE-P Current-Phase)
(6)                       (RESET-ROLE Current-Observation)
(7)                       (CALL-IA Select-Current-Observation)
(8)                       (COND ((ROLE-P Current-Observation)
(9)                              (CALL-IA Make-Observation)
(10)                             (CALL-IA Infer-Environment-Data)
(11)                             (CALL-IA Specify-Aphase-Adaptation)))
(12)                      (CALL-IA Select-Noncurrent-Phase)
(13)                      (COND ((ROLE-P Current-Phase)
(14)                             (RESET-ROLE Current-Action)
(15)                             (CALL-IA Select-Current-Action)
(16)                             (CALL-IA Specify-Action-Adaptation))
(17)                     (T (RETURN)))))
(18)         (T (RETURN))))
```

equivalent LISP iteration statements). However, the statements such as RESET-ROLE current-observation (see Line 6 of Table 8) are new. This is due to a sloppy aspect of conceptual modeling. Even though the current metaclass assignment in a KADS model is part of the state of the problem solver, it is not modeled with sufficient rigor to transform it straight away into an operational model. The statement RESET-ROLE current-observation accounts for the fact that this role assignment is dynamic and is computed repeatedly, every time the system proposes new laboratory tests for the current day of the therapy.

5.3 Which Changes did Occurr?

This section recapitulates and discusses the changes that occurred in the transition from K-ONCOCIN to O-ONCOCIN. We show that there were no significant modifications to the knowledge-level aspects of the model. There was much change in terminology and syntax, but little structural modification. We observed the following changes:

1. In the inference structure (see Figure 5 for the KADS version and Figure 7 for the OMOS one) we added the inference action OBTAIN-OBSERVATION to account for the corresponding KADS task.

2. The data flow in the inference structure was specified in more detail (i.e., we distinguished between input, control, and output), and some necessary data flow information was added to the diagram (e.g., SELECT-NEXT-PHASE uses the role obsolete-phase).

3. The inference action INITIATE-CURRENT-PHASE was added for computational reasons. We must initialize the counter for the current days in the therapy before starting treatment recommendations. This is a point where representational concerns are introduced into the model.

4. When transforming the KADS task structure (see Table 8) into an OMOS control structure (see Table 11) the iterative structure could be maintained. However, RESET-ROLE–statements had to be added to account for the temporal nature of certain role assignments in this cyclic process. KADS conceptual models do not account for such phenomena.

5. In the domain model we had to account for the necessity to distinguish between different occurrences of chemotherapies (see Figure 6). This resulted in a straightforward extension of the KADS domain model (see Figure 4), and did not require structural modifications.

Thus, the only real change to the model that went beyond additional specification of knowledge to disambiguate the model was the definition of the inference action INITIATE-CURRENT-PHASE. This is clearly a representational concern, showing up in a place of the model that one would like to reflect knowledge-level concerns only.

5.4 The OMOS Model of the Task

The model of the task consists of the model of the domain combined with the model of the method. The development cycle model for OMOS (see [Linster, 1992a]) states that the structures of the models are developed first; then the different models are combined into a model of the task, which then is the framework for the acquisition of the detailed knowledge. Thus, we must differentiate between the *minimal model of the task* and the *full-blown model of the task.* (see Figure 1). The first one being the prerequisite for guided knowledge acquisition; the second one being the result of the knowledge acquisition process. In a minimal task model, the domain model contains only those frames and relations that are used directly in the mapping from the method to the domain. Tuples or relations, instances or sub-frames can be gradually added to the minimal model, to turn it into a full-blown model. Additional sub-structures (frames or relations) do not introduce distinctions that the method uses. Thus, they are not part of the minimal task model.
This distinction is important for two reasons. First, the minimal task model contains those domain distinctions that are accessed directly by the problem-solving method. This makes it an abstract re-usable framework that can be used if a similar problem is to be solved again in the same domain.
Second, the minimal task model provides a framework in which one can analyze the use of domain knowledge elements by the problem-solving method. Thus, whereas the minimal task model must be developed manually, the extension of the minimal task model into a full-blown task model is supported by automated analysis tools.
OMOS provides several tools to analyze the use of the domain knowledge in the framework of the task model. The *dead-end analysis* looks for intermediary role-assignments that are not used by any of the subsequent inferences. For example in Table 12, the analysis determines that the domain concept *chest-x-ray* is not used in the reasoning process. It proposes to modify the relation `test-due`, so that the inference action SELECT-CURRENT-OBSERVATION disposes of the necessary detailed knowledge to consider *chest-x-ray* in its inferences. Other tools focus on the definition of domain relations (coverage analysis) and inference actions (loop-hole analysis).

5.5 Summary

This section illustrated how a KADS model of expertise—a conceptual, non-operational model—is transformed into an operational OMOS model. We explained the details of

Table 12: The diagnostic text produced by a dead-end analysis of the role available-observation. The analysis determines that the domain concept *chest-x-ray* is not used in the reasoning process. It proposes to modify the relation **test-due**, so that the inference action SELECT-CURRENT-OBSERVATION disposes of the necessary detailed knowledge to consider *chest-x-ray* in its inferences.

```
The domain concept CHEST-XRAY can be assigned to the role
AVAILABLE-OBSERVATION, but it is not used by
the inference action SELECT-CURRENT-OBSERVATION
which uses AVAILABLE-OBSERVATION as control- or as input-role.

To change this do AT LEAST ONE of the following:

     Add a tuple to the relation TEST-DUE, which is
     used by the inference action SELECT-CURRENT-OBSERVATION.
     CHEST-XRAY must be mentioned in the 0. argument of the
     tupel, which is of type (INSTANCE TEST).
     ...
```

this transformation process and showed that it requires only minor modifications to the knowledge-level aspects of the model of expertise. However the new representation provides considerable information for the continuing extension and disambiguation of the model of expertise.

6 Pros and Cons of Using OMOS for KADS

This section discusses on the one side the use of OMOS to implement KADS conceptual models, and on the other side the more general interaction principles between conceptual modeling in a pre-formal phase and computer-supported model elaboration after the phases of conceptual modeling.

Besides the implementation of the cancer-chemotherapy administration task, OMOS has been used to implement a KADS model for clamping tool selection [Kühn *et al.*, 1991]. Both model developments showed that in the early, pre-formal phases of knowledge-based systems development one needs the flexibility of a conceptual modeling approach à la KADS. In these phases the support for conferring meaning to new situations, for construing them in a modeling vocabulary and for communicating an understanding of a situation to other partners participating in the modeling process is far more important than formal analysis or executability. However, as soon as the structures that result from the process of conceptual modeling start to stabilize and become coherent, than the advantages of formal analysis and executability outweigh conceptual modeling.

For the modeling of the cancer-chemotherapy administration task this means the following: a formal modeling environment (e.g., KL-ONE) could not have provided the necessary "sloppiness" for the initial process of making sense, as documented in [Linster and Musen, 1991; Linster and Musen, 1992]. However, as soon as the model of the task has been completed, the elaboration of the detailed knowledge can be supported by automated tools working in the framework of the model of the task. These tools help considerably in completing the domain knowledge by providing diagnostic messages such as the one rendered in Table 12. Furthermore, the fact that one has to disambiguate every inference in the reasoning process provides additional information about the modeling structures. For example, the KADS conceptual model does not document an information flow between the knowledge source SELECT-NEXT-

PHASE and the metaclass obsolete-phase (Figure 5), but when we implemented the model, it became obvious that to select the next phase, one must know about past phases (Figure 7). The refinement of the representation of chemotherapies (discussed in Section 5.2.1) is another example that shows that a knowledge engineer needs computer support relying on formal properties to handle the sheer complexity of a larger model. Theoretically, this ambiguity could have been detected in a paper knowledge base, but it was uncovered only when executing the model. These examples should make it clear that one needs both conceptual and formal modeling. KADS-like approaches and OMOS-like approaches are complementary in the development of models. Each approach has its own raison d'être, KADS in the early model-creating phases, OMOS in the phases of model elaboration and extension.

Acknowledgement

I am very grateful to Mark Musen for developing the K-ONCOCIN model together with me, and for the worthy discussions that we had about modeling problem-solving methods. Many of the ideas presented in this paper result from discussions with Wolfgang Gräther, Werner Karbach, Angi Voß, Jürgen Walther, and the other members of the knowledge modeling team at GMD. Sonja Branskat and three anonymous reviewers commented on earlier versions of this paper.

References

[Boose, 1985] John H. Boose. A knowledge acquisition program for expert systems based on personal construct psychology. *International Journal of Man-Machine Studies*, 23:495 – 525, 1985.

[Brachman, 1979] Ronald J. Brachman. On the epistemological status of semantic networks. In N.V. Findler, editor, *Associative Networks: Representation and Use of Knowledge by Computers*, pages 3 – 50 50. Academic Press, New York, 1979.

[Chandrasekaran, 1988] B. Chandrasekaran. Generic tasks as building blocks for knowledge-based systems: The diagnosis and routine design examples. *The Knowledge Engineering Review*, 3(3):183 – 210, Octobre 1988.

[Christaller *et al.*, 1992] Thomas Christaller, Franco di Primio, and Angi Voß. *The AI workbench BABYLON*. Academic Press, London, 1992.

[Clancey, 1989] William J. Clancey. The knowledge level reinterpreted: Modeling how systems interact. *Machine Learning, Special Issue on Knowledge Acquisition*, 4(3, 4):285 – 292, 1989.

[Diederich *et al.*, 1987] Joachim Diederich, Ingo Ruhmann, and Mark May. KRITON: A knowledge acquisition tool for expert systems. *International Journal of Man-Machine Studies*, 26:29 – 40, 1987.

[Friedland and Iwasaki, 1985] Peter E. Friedland and Yumi Iwasaki. The concept and implementation of skeletal plans. *Journal of Automated Reasoning*, pages 161–208, 1985.

[Karbach *et al.*, 1991] Werner Karbach, Angi Voß, Ralf Schukey, and Uwe Drouven. MODEL-K: Prototyping at the knowledge level. In *Proceedings of the First International Conference on Knowledge Modeling and Expertise Transfer*, Sophia Antipolis, France, 1991.

[Kühn *et al.*, 1991] Otto Kühn, Marc Linster, and Gabi Schmidt. Clamping, COKAM, KADS and OMOS. In *Proceedings of EKAW91*. University of Strathclyde, 1991. Also published as Technical Memo TM-91-03 of DFKI, Kaiserslautern.

[Linster and Musen, 1991] Marc Linster and Mark Musen. Using KADS to build K-ONCOCIN: A conceptual model of ONCOCIN. In John H. Boose and Brian R. Gaines, editors, *Proceedings of the 6th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, pages 13/1 – 13/20, Calgary, 1991. AAAI, University of Calgary.

[Linster and Musen, 1992] Marc Linster and Mark Musen. Use of KADS to create a conceptual model of the ONCOCIN task. *Knowledge Acquisition*, 4(1):55 – 88, 1992.

[Linster, 1989] Marc Linster. Towards a second generation knowledge acquisition tool. *Knowledge Acquisition*, 1(2):163 – 183, 1989.

[Linster, 1992a] Marc Linster. Explicit and operational models as a basis for second generation knowledge-acquisition tools. In Jean-Marc David, Jean-Paul Krivine, and Reid Simmons, editors, *Second generation expert systems*. Springer, 1992. Forthcoming.

[Linster, 1992b] Marc Linster. *Knowledge acquisition based on explicit methods of problem-solving*. PhD thesis, University of Kaiserslautern, Kaiserslautern, February 1992.

[Linster, 1992c] Marc Linster. Linking modeling to make sense and modeling to implement systems in an operational environment. In Thomas Wetter, Klaus-Dieter Althoff, John Boose, Brian Gaines, Marc Linster, and Franz Schmalhofer, editors, *Current developments in knowledge acquisition: EKAW92*, volume 599 of *LNAI*, pages 55 – 74, Heidelberg, 1992. GI, ECCAI, Springer-Verlag.

[Linster, 1992d] Marc Linster. Using the operational modeling language OMOS to tackle the Sisyphus'92 office-planning problem. In Marc Linster, editor, *Sisyphus'92: Models of problem solving*, volume 630 of *Technical report of GMD*, St. Augustin, 1992. GMD.

[Morik, 1989] Katharina Morik. Sloppy modeling. In Joerg Siekmann, editor, *Knowledge Representation and Organization in Machine Learning*, volume 347 of *LNAI*, pages 107 – 133. Springer Verlag, Heidelberg, 1989.

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

[Wielinga *et al.*, 1992] Bob Wielinga, Guus Schreiber, and Jost Breuker. KADS: A modelling approach to knowledge engineering. *Knowledge Acquisition*, 4(1):5 – 54, 1992.

[Wrobel, 1988] Stefan Wrobel. Design goals for sloppy modeling systems. *International Journal of Man Machine Systems*, 24(4):461 – 477, October 1988.

Increasingly Automated Procedure Acquisition In Dynamic Systems

Nathalie Mathé[1] & Smadar T. Kedar[2]
NASA Ames Research Center
Mail Stop 269-2
Moffett Field, CA 94035, USA
mathe@ptolemy.arc.nasa.gov
kedar@ptolemy.arc.nasa.gov

ABSTRACT

Procedures are widely used by operators for controlling complex dynamic systems. Currently, most development of such procedures is done manually, consuming a large amount of paper, time and manpower in the process. While automated knowledge acquisition is an active field of research, not much attention has been paid to the problem of computer-assisted acquisition and refinement of complex procedures for dynamic systems. This paper presents the Procedure Acquisition for Reactive Control Assistant (PARC), which is designed to assist users in more systematically and automatically encoding and refining complex procedures. PARC is able to elicit knowledge interactively from the user during operation of the dynamic system. We categorize procedure refinement into two stages : *diagnosis* - diagnose the failure and choose a repair - and *repair* - plan and perform the repair. The basic approach taken in PARC is to assist the user in all steps of this process by providing increased levels of assistance with layered tools. We illustrate the operation of PARC in refining procedures for the control of a robot arm.

1. INTRODUCTION

Procedures are widely used by operators for controlling complex dynamic systems such as airplanes, nuclear power plants, and the Space Shuttle. Procedures specify how to conduct a set of predetermined sub-tasks (or actions) that are components of a higher level task. For example, they dictate the manner by which a pilot is expected to interact with an aircraft's systems. Currently, most development of such procedures is done manually, consuming a large amount of paper, time and manpower in the process (Mathé, 1990b; Degani *et al.*, 1991). First, a system designer encodes nominal procedures which may be incomplete and incorrect. Then, during the course of system testing and operation, if the procedures fail to perform as expected, the user needs to cope with the failure and find ways to recover. Detecting and correcting the procedures deemed responsible for the failure is done later, off-line. Such procedure refinement never ceases -- it is done incrementally over the lifetime of the dynamic system.

There is a need for a tool to assist users in encoding procedures, and refining them during operations. While automated knowledge acquisition is an active field of research, not much

---

[1]Dr. Mathé is a European Space Agency fellow and currently a NASA-Ames Research Associate.

[2]Dr. Kedar is a contractor at Sterling Software, Inc., and is currently a visiting researcher at the Institute for the Learning Sciences, Northwestern University.

attention has been paid to the problem of acquiring and refining complex procedures for dynamic systems. Most attempts at procedure acquisition to date have proposed advanced editors, including flow-chart diagram editors, or other similar graphical editors (Saito *et al.*, 1991). Other attempts have dealt with procedure acquisition in static domains such as medical diagnosis (Musen *et al.*, 1987). While those previous studies mainly addressed the elicitation phase of procedure acquisition[3], we focus in this work on the refinement phase. Research in machine learning addresses procedure elicitation and refinement by providing fully automated methods (Ourston & Mooney, 1990; Gil, 1991; Kedar, 1991). However, these methods rely on strong assumptions about the domain (e.g. complete perception and prediction), which are violated in most real-world environments.

The Procedure Acquisition for Reactive Control assistant (PARC) is designed to assist users in more systematically and automatically encoding and refining complex procedures for dynamic systems. Thus it is expected to increase the accuracy of procedures with decreased time and effort on the part of the user. In order to be effective, we require that it assist in all stages of the procedure refinement process.

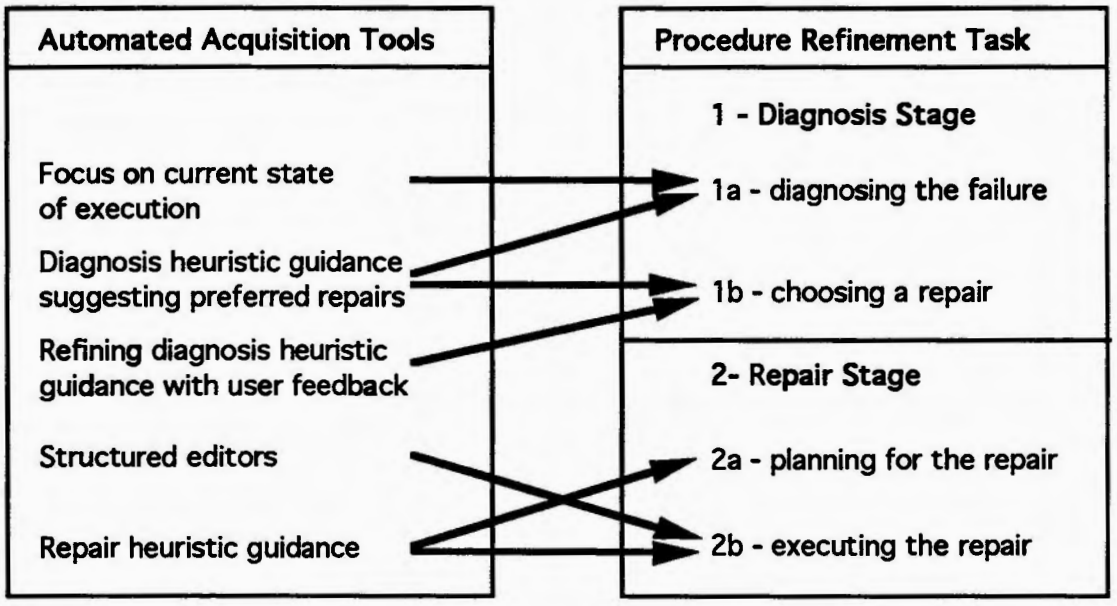


Figure 1: Assistance levels supporting the procedure refinement stages

Given the incremental nature of procedure refinement, PARC elicits knowledge during operation of the dynamic system. During the control phase, the user supervises the dynamic system using procedures suggested by a control assistant. When the user disagrees with the control assistant (such an event is called a *failure*), PARC starts a procedure acquisition dialog. We categorize the procedure refinement process into two stages:

(1) **Diagnosis stage**: (1a) the user diagnoses the failure
and (1b) chooses the repair needed to correct the procedure base;

(2) **Repair stage**: (2a) the user plans for the repair
and (2b) performs the repair.

This refinement process is incremental and continues throughout further execution.

---

[3]Knowledge Acquisition can be broadly classified into three phases: elicitation, refinement and reformulation (Bareiss *et. al.*, 1989).

PARC is designed with layered KA tools, providing increased automated assistance to support more steps of the refinement task (Figure 1). At the least automated level of assistance, PARC provides information on the control assistant's current state of execution, assisting the user in diagnosing the failure. It also provides structured editors that can be used stand-alone to perform the repair. Secondly, PARC provides more automatic heuristic guidance in both stages. In particular, it suggests which types of refinements are likely to correct which types of failures, assisting both steps of the diagnosis stage. It also leads the user through the repair stage by providing heuristic guidance in planning and executing the repair. Finally, PARC provides automatic reinforcement of the heuristic diagnosis guidance. It reinforces the heuristics with user feedback in order to refine and improve its guidance over time.

In the next section, we will present an overview of the procedure acquisition assistant, including a brief procedure acquisition example, descriptions of the performance system and of the procedure acquisition assistant, and of their associated knowledge representation for procedures. In the third and fourth sections, we describe the procedure acquisition assistant with a detailed running example. We analyze the procedure acquisition assistant in the fifth section and conclude by presenting some related work and future directions of research.

2. OVERVIEW OF THE PROCEDURE ACQUISITION ASSISTANT

2.1. A Sample Procedure Acquisition Dialog

In this section we illustrate PARC's operation using a simple example from a robotics domain. in our robotics lab we use a Puma 560 robot has seven degrees of freedom, a gripper, and several sensors. A sonar sensor measures the distance to the closest object in the environment. A force-torque sensor is used to detect obstacles and perform several types of commands. A camera attached on the gripper is used with image analysis software to detect cubes and compute their positions in the environment. The user has no direct view of the environment. A video screen displays a global view of the robot environment from a fixed camera. The user controls the robot through a graphical interface which includes graphical sensors values displays, a 3D graphical simulator, and a high level command language with menus and graphical command panels (Figure 2). Messages and procedures are also displayed on the interface by the control assistant.

In this example, the task of the user is to build a structure of cubes and beams with the robotic arm. The control assistant suggests executing the sub-task "Approach Cube" using a *guarded move* robot command. A guarded move uses the force-torque sensor to detect any collision during the motion. The motion is stopped when the goal position is reached (successful command execution) or when an obstacle is detected (command execution failure). The user agrees to execute the "Approach Cube" procedure and observes the control assistant perform the guarded move. Part way through the guarded move, the assistant abandons it to select an "Avoid Obstacle" procedure, trying to avoid a nonexistent obstacle. At this point, the user notifies the assistant that she disagrees; she did not expect the "Avoid Obstacle" procedure in the midst of the "Approach Cube" procedure, because she doesn't see any obstacle on the video screen.

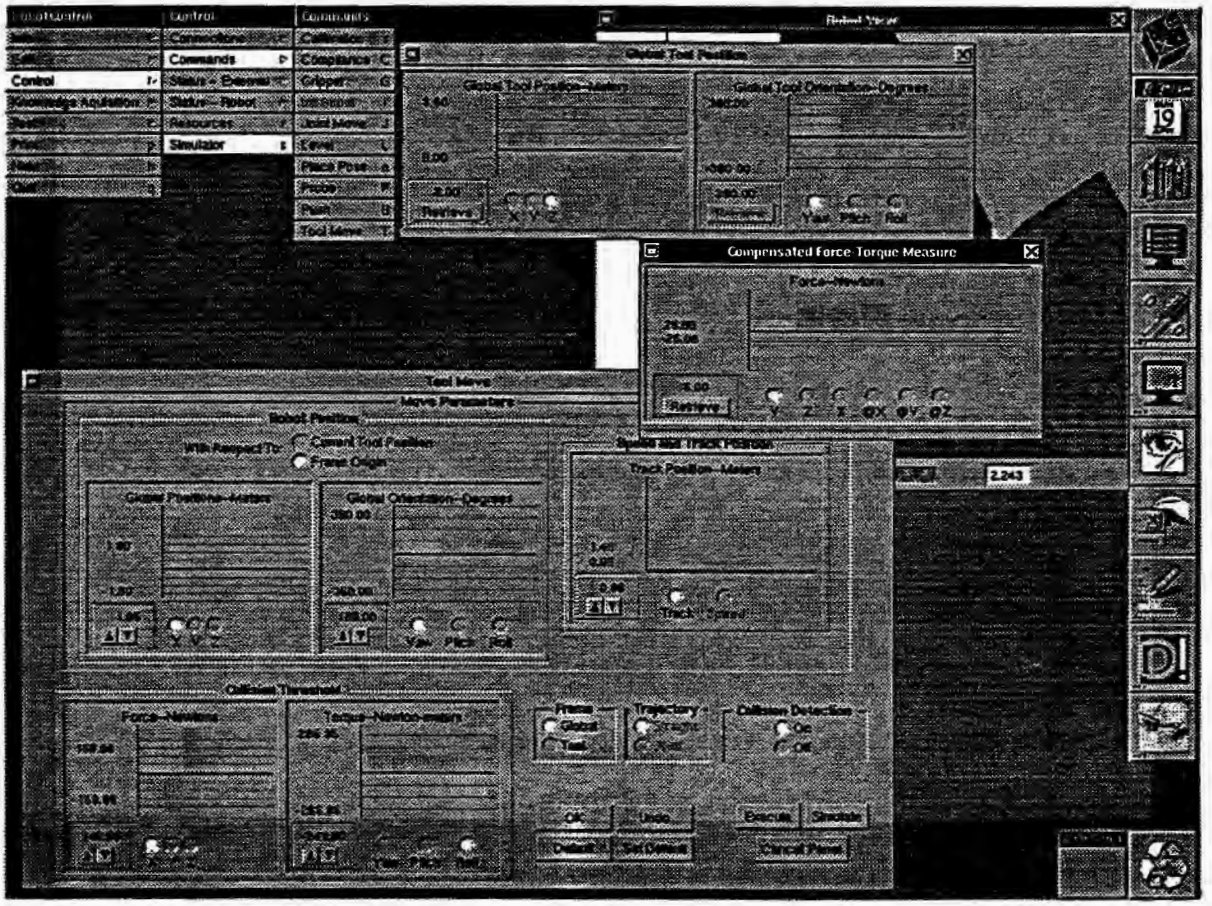


Figure 2: User Interface for Robotic Control

Given this failure[4], a procedure refinement dialog begins. First, the procedure acquisition assistant provides to the user the system's current state of execution: the procedure for "Approach Cube" is followed by the procedure "Avoid Obstacle" triggered by the abnormal outcome "obstacle". PARC assists the user in diagnosing the failure: there is only one abnormal outcome for the "Approach Cube" procedure, such that any execution failure feedback would be assumed to be an obstacle, and thus to be avoided by invoking the "Avoid Obstacle" procedure. In fact, one of the other reasons for the execution failure of the guarded move command is if the force-torque sensor stops working. The event corresponding to a sensor breakdown has been displayed to the user, and by checking the sensors status, the user discovers that this is the problem. PARC then assists the user is in choosing an appropriate repair for the procedure. Given all possible repairs (adding or removing any preconditions, outcomes, or procedure links), the heuristic guidance suggests that for this type of failure, an additional abnormal outcome and associated recovery procedure is needed. The user selects this repair and PARC then presents a dialog which leads her through the steps of the repair. Later on in system operation, the user finds that once again, when the force torque sensor was not working, the control assistant suggests the "Avoid Obstacle" procedure. By diagnosing the failure, the user notes that the heuristic guidance was incomplete, and that the criticality order of the two abnormal outcomes needed to be modified as well The user's feedback on the repair choice is used to modify and reinforce the corrected heuristic.

---

[4]Note that a command execution failure is different from what we call a failure: the fact that the user disagrees with the control assistant. A failure is not always associated to an unexpected command execution failure, like in this example. It may be due to an unexpected situation for which the control assistant has no knowledge to deal with.

2.2. The Architecture

The architecture consists of two main components: the *performance system* and the *knowledge acquisition system* (Figure 3). The task of the performance system is to assist the user in controlling a dynamic system (e.g. a robot, an airplane). It is thus called a control assistant. In our case, the *task domain* is robotic arm control. The KA system is the procedure acquisition assistant, PARC. The procedure acquisition dialog is triggered when the user disagrees with the control assistant. The procedure acquisition system uses the dynamic problem-solving ability of the performance system to support procedure refinement. PARC operates as follows: given the current procedure base and set of active procedures, it assists the user in (i) diagnosing the failure; and (ii) repairing the failure. Once the failure has been repaired (or if the user agrees with the control assistant), the control assistant resumes execution. In the following sections we will briefly describe each of these aspects.

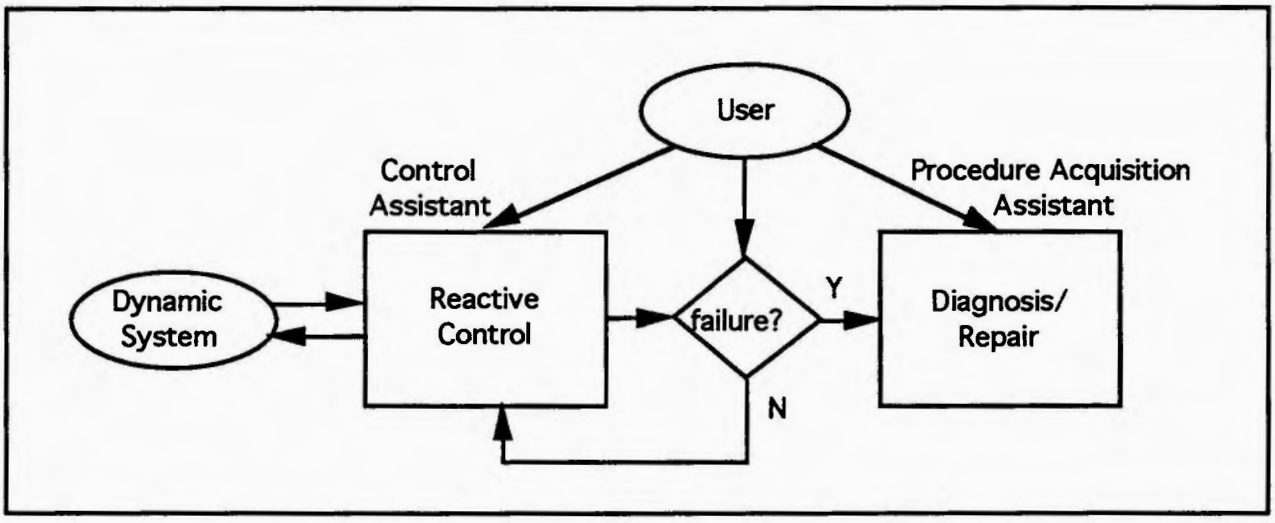


Figure 3: Architecture of the Control and Procedure Acquisition Assistants

2.2.1. The Performance System : For assisting the user in controlling the robotic arm, the control assistant's *problem solving method* is reactive control: given the task goal and the set of active procedures, analyze the current situation, select the next appropriate procedure, execute it and monitor its results (Mathé, 1990a). Procedures are represented as *Blocks*, a formalism described in section 2.2.3., and the control assistant's problem solving method is based on the inference mechanism associated with the block representation (Boy, 1989; Mathé, 1990b-c).

The control assistant operates as follows. It monitors the environment status, the robotic arm status and user's actions on the interface. When an event is perceived, it is interpreted by the assistant with respect to its current expectations. The assistant then decides if the current procedure still applies or if another procedure should be selected. In the latter case, it displays a message to the user explaining the new situation and the name of the selected procedure. If the user agrees with the assistant's selection, the selected procedure is displayed on the user interface (by selecting the appropriate display and setting parameters values - Figure 2). This is possible since procedures are expressed in terms of user's actions on the interface (like flight procedures are expressed in terms of pilot's actions in the cockpit), and not directly in terms of robotic instructions common for most autonomous control systems. The user may then choose to execute this procedure, modify some parameters values before executing it, or ignore it and select another command panel. The control assistant then executes the procedure by translating it into low level instructions for the robotic arm and monitoring its results.

2.2.2. The Procedure Acquisition System : PARC *acquires knowledge* that is incorporated as extensions and refinements to an initial skeletal procedure base. This initial procedure base, as well as models of the dynamic system and its environment, is acquired manually during a preliminary elicitation phase with the domain designers. PARC assists the user in incrementally refining this procedure base.

PARC provides tools automating each step of the refinement process. Once the user detects a failure, it assists her in diagnosing why the control assistant suggested the wrong procedure, by displaying information on the set of active procedures involved in the current execution state. Then, PARC assists the user in choosing the repair needed to correct the faulty procedures by suggesting a subset of preferred repairs among all possible repair options, based on some heuristic knowledge. This heuristic guidance can be automatically tailored with user feedback in order to reinforce particular preferred repairs, suggest new repairs, or include domain-specific knowledge. Finally, for assisting the user in planning and performing the repair, PARC proposes a procedure acquisition dialog driven by some heuristic procedural knowledge. This includes knowledge on how to decompose a particular repair into a sequence of steps, on how to take into account the current execution state, and on how to appropriately use the structured editors. Those structured editors provide direct access to procedures and are based on the knowledge representation for procedures described below.

2.2.3. Procedure Representation : Procedures in the control assistant are represented as *Blocks*, a representation designed to facilitate incremental procedure development. In a previous robotic application where procedures were manually acquired through experimental protocols, we developed a cognitive model of reactive control. Following this study, we designed the block representation and its associated inference mechanism as an appropriate formalism for this model (Mathé, 1990b-c). The basic structure of this formalism is a *block* of knowledge. A procedure base is represented as a network of blocks organized in several hierarchical levels. The inference mechanism associated with this formalism is independent of the content of the procedure base. Different procedure bases can be loaded into the system, depending on the particular application task involved. A block includes the following components:

(1) name;
(2) hierarchical level;
(3) list of preconditions;
(4) list of actions;
(4) list of goals with their lists of associated blocks; and
(5) list of abnormal conditions with their lists of associated blocks.

A block is graphically represented as in Figure 4. It is divided into four parts corresponding to its preconditions, actions, goals and abnormal conditions. The arrows represent the links towards the blocks associated to goals and abnormal conditions. Blocks of lower hierarchical level have a grey background, and block of same or higher hierarchical level have a white background.

Actions are either elementary actions (executable on the user interface) or sub-blocks of lower hierarchical level. Goals and abnormal conditions are terminal conditions which are tested during and after the execution of the block actions. For a given block, each goal and abnormal condition is linked to a list of blocks which contains possible procedures to be executed next if the goal or abnormal condition is satisfied. Abnormal conditions and goals differ only semantically: goals correspond to a successful execution; abnormal conditions correspond to all the abnormal situations which may occur during or after execution. Their associated blocks represent how to recover from a particular abnormal situation.

Depending on its complexity, a procedure may be represented by a single block, a hierarchy of blocks or a network of blocks. A simple procedure is represented by a single block whose actions are elementary. A more complex procedure may be represented by a block whose actions are sub-blocks at a lower hierarchical level, containing themselves either elementary actions or lower level

sub-blocks. Finally, a set of complex procedures is represented by a network of blocks, hierarchically organized and transversely connected through their goals and abnormal conditions. Given the procedure hierarchy, the context of a block is defined as the set of its parent blocks' preconditions. The context structure and hence the hierarchical decomposition of blocks is determined by task analysis. This representation supports incremental procedure development through acquisition of new blocks and refinement of existing blocks by adding abnormal conditions and associated recovery procedures and by learning contextual conditions (Boy, 1989).

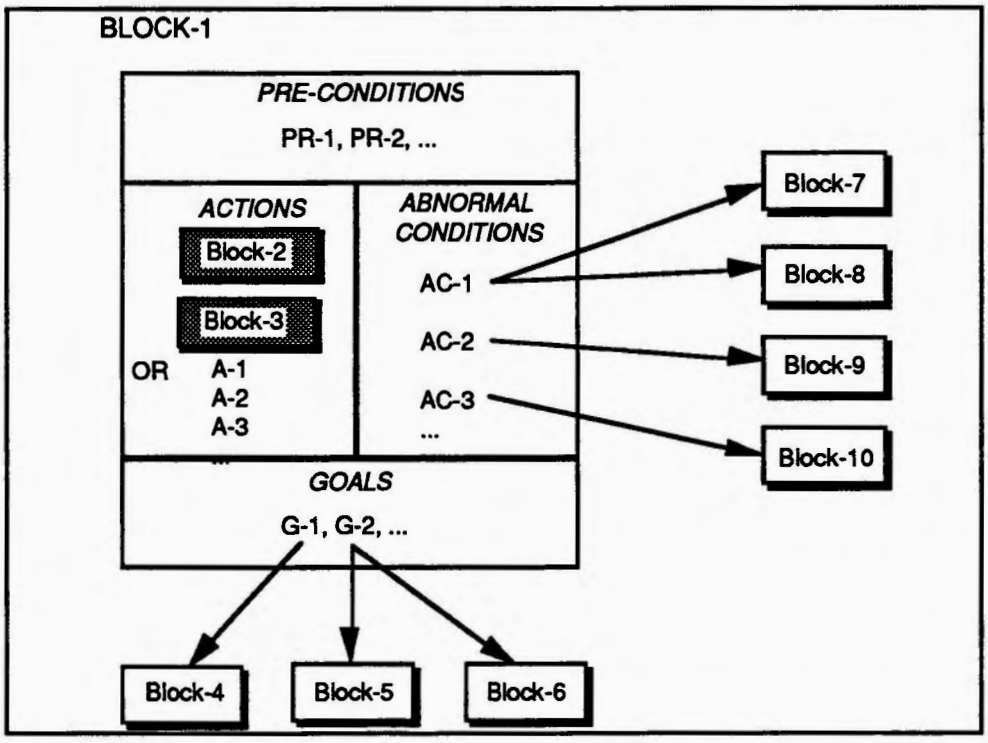


Figure 4: Block graphical representation

The behavior of the control assistant is based on the inference mechanism associated with the block representation:

(1) Sensory inputs from the environment, the robot and the user interface are matched with the system current expectations. Expectations correspond to preconditions of blocks selected for execution, and to goals and abnormal conditions of active blocks.

(2) The most critical of the best matched expectations is selected by the assistant.

(3) Depending on the type of the selected expectation, the block to which it belongs and the next blocks associated to it, the assistant determines which procedures become active, inactive or selected for execution. If a low level block becomes active, its actions are executed on the user interface. The set of of current expectations is updated and used during the next pattern-matching phase.

2.2.4. Example of Procedures for Robotic Control : Using this representation, we have encoded in the control assistant the first set of nominal procedures corresponding to the cubes and beams assembly task. We have manually acquired this set of procedures through informal interviews with our robotics expert. For example, the task "Insert Cube in Structure at Destination Pose" is decomposed into several phases: "Choose Cube," "Pick up Cube," "Move Cube to Destination Pose," and "Fasten Cube into Position" which represent a nominal sequence of procedures linked through their goals. The sub-phase "Approach Cube" (Figure 5), part of "Pick up Cube," has one goal "gripper at position above cube" linked to the next nominal procedure "Align Gripper," and

one abnormal conditions "obstacle" linked to the next alternative recovery procedures "Find Collision free Path to Goal Position" and "Avoid Obstacle".

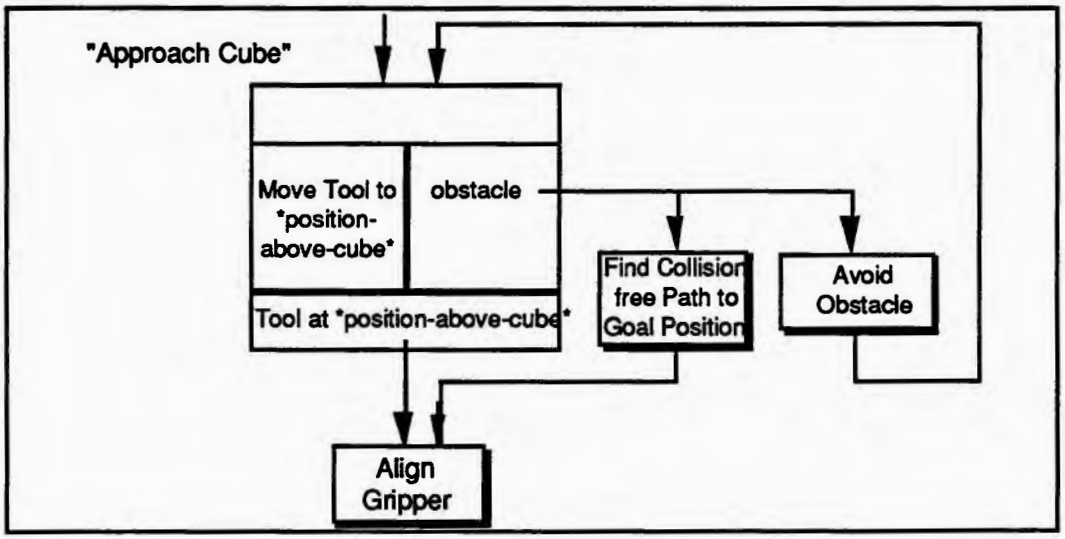


Figure 5: Part of the "Pick up Cube" procedure

3. AUTOMATING THE DIAGNOSIS STAGE

The procedure acquisition dialog is triggered when the user disagrees with the control assistant. We called such an event a failure. In the following sections we describe the forms of assistance provided to the user in diagnosing the failure.

3.1. Detecting a Failure

During reactive control, the control assistant analyzes the situation, matching it with its current expectations (preconditions, goals, and abnormal conditions). It then decides if the current procedures still apply or if another procedure should be suggested to the user. In the latter case, it displays a message, as shown in Figure 6, informing the user about: (1) perceived events (if any); (2) the recognized expected condition; and (3) the suggested procedure or the result of a previously active procedure (normal or abnormal). The user may agree on the recognized situation and the suggested procedure ("OK" button in Figure 6), in which case the execution will continue (cf. Section 2); or the user may disagree and start a procedure acquisition dialog by accessing the other choices considered by the control assistant ("See Other Choices" button in Figure 6).

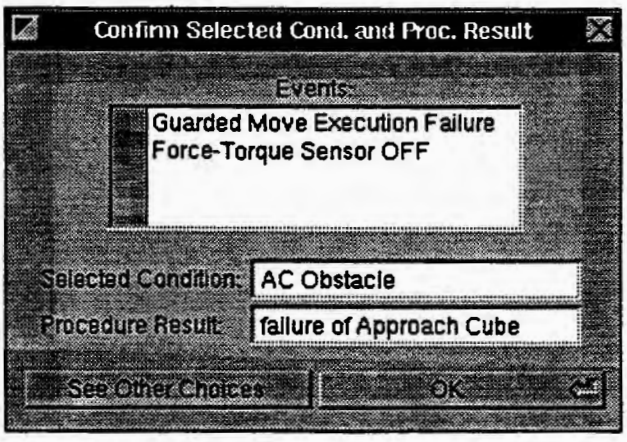


Figure 6: Confirm Selection acquisition panel

Taking the example described in Section 2, the user wants the robotic arm to insert the next cube into the partially built structure. In order to do so, the control assistant has selected a cube on the table using the "Choose Cube" procedure, then proposed to the user that she executes the "Approach Cube" procedure. This procedure uses a guarded move robot command. The user agrees on this procedure suggestion and observes the robot perform the guarded move. Part way through the guarded move, the control assistant informs the user that the "Approach Cube" procedure has failed (Figure 6). The events "execution failure" and "force-torque sensor off" have been received by the control assistant and the abnormal condition "obstacle" recognized. Since the user doesn't see on the video screen any obstacle in front of the robot, she notifies the assistant that she disagrees on the recognized situation, thus detecting a failure.

3.2. Diagnosing the Failure

Once the user disagrees with the control assistant, the first step of the refinement process is to diagnose the failure. The diagnosis consists in determining what is wrong in the suggestion of the control assistant, given the current situation in the environment. For example, no procedure may have been selected if it is missing in the procedure base; or several expectations may have been recognized, but the wrong one selected.

To help the user diagnose the failure, PARC provides information on the assistant's current state of execution. It displays the Situation Description panel (Figure 7) which focuses on portions of the procedure base that are involved in the current execution. This panel describes the current state of execution in a scrollable tree format: current active procedures, current expectations, and next procedures linked to these expectations. The first column displays the names of active procedures in hierarchical order. The second column displays current expectations associated to each active procedure, with their name, type (precondition, goal or abnormal condition), degree of match and degree of criticality. The third column displays the names of procedures to be executed next, associated with each expected condition. It shows the user which procedures may become active next depending on the condition she selects. By default, the condition selected by the control assistant is highlighted when the panel is displayed.

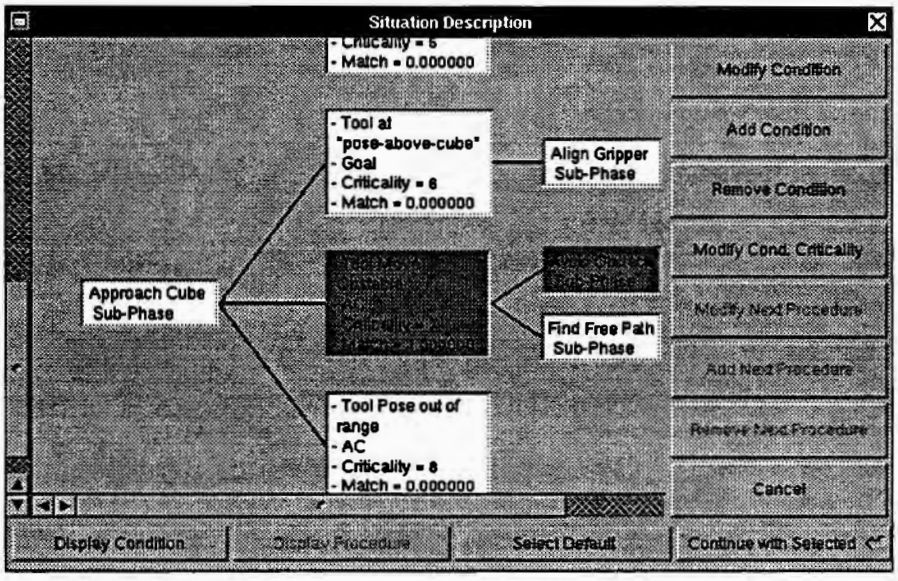


Figure 7: Situation Description acquisition panel

In our example, PARC displays the set of active procedures in a scrollable tree format: "Build Structure," "Insert Cube in Structure at Destination Pose," "Pick up Cube," and "Approach Cube" at the lowest hierarchical level. Figure 7 shows the bottom part of this tree, including

the "Approach Cube" procedure in the first column. In the second column, the expected condition "obstacle" selected by the control assistant is highlighted. It is an abnormal condition of the "Approach Cube" procedure. The procedure "Avoid Obstacle" is going to be executed if the user does not change the selection and repair the procedure base. By accessing the "obstacle" condition, the user sees that the "execution failure" event can only be interpreted as an obstacle by the system. In fact, given her knowledge of the domain, she knows that one of the other reasons for the execution failure of the guarded move command is if the force-torque sensor stops working. When selecting the "obstacle" condition, the control assistant has displayed to the user the event corresponding to the sensor breakdown. By checking the sensors status, she decides that this is the problem. Thus, the user diagnoses that the "obstacle" condition has been incorrectly selected and that the sensor breakdown event should have be recognized instead.

3.3. Choosing a Repair

After diagnosing the failure, the user may select a repair or directly access the editors. In the latter case, she has to rely on her own knowledge for choosing, planning and performing the repair. In this section, we describe the assistance provided by PARC for helping the user choose the repair.

First, the set of all possible repair options is displayed on the Situation Description panel (list of buttons on the right hand side - Figure 7): modify, add or remove an expected condition; modify an expected condition criticality; modify, add or remove a next procedure. Those options are not exhaustive nor exclusive, but found to be the most useful for recovering from a failure[5].

Next, PARC assists the user choose the repair among all possible repair options by suggesting a subset of preferred repairs and highlighting the corresponding buttons. Heuristics are used to determine the subset of preferred repairs given the failure type. This heuristic guidance encodes knowledge from the knowledge engineer. The types of failures currently identified and their associated preferred repairs are given in Table 1. The type of a failure is determined by PARC using the current state of execution and the pattern matching result. It does not depend on any domain knowledge.

We illustrate two simple cases from Table 1. In case (3), if none of the expected conditions matches a perceived event (a *surprise*) and if this event is judged significant by the user, PARC suggests that the user add a new expected condition describing this event and an associated procedure specifying how to recover from it, or to modify an existing expected condition. In case (7), if the recognized expected conditions are correct but the wrong one was suggested by the control assistant, PARC suggests that the user increases the degree of criticality of the right condition (so that it becomes more critical than all the other ones recognized), in order for this condition to be correctly selected the next time and in the same situation. If the user does not modify the criticality order before selecting the right condition, the procedure base is not modified and the same failure may happen again.

In our example, the control assistant has selected the wrong abnormal condition "obstacle" due to the sensor breakdown. It corresponds to case (6) in the table. The heuristic guidance suggests that for this type of failure, an additional abnormal condition describing the unexpected event (force-torque sensor not working) is needed, with its associated recovery procedure. The user selects this repair and the procedure acquisition assistant presents a dialog which leads her through the steps of the repair (cf Section 4).

---

[5]This is a first approximation of which repair actions would be needed. We plan to provide more help to the user by suggesting specific procedure components on which those actions should be applied. To keep this paper clear, we have only given examples where one repair option is needed to repair a failure, but repairs including several repair options are also supported by PARC.

Failures Types		Preferred Repairs
No Match	*No procedure associated to selected goal or abnormal condition* 1	-> add next procedure to selected goal or abnormal condition
	No Surprise 2	-> add expected condition -> modify expected condition -> add next procedure
	Surprise 3	-> add expected condition -> modify expected condition
One Match	*General* 4	-> modify the selected expected condition -> add expected condition
	If Abnormal Condition selected and there is no uninterpreted event 5	-> add a new abnormal condition and associated recovery procedure and modify the one selected -> add a new abnormal condition and associated recovery procedure
	If Abnormal Condition selected and there is an uninterpreted event 6	-> add abnormal condition corresponding to the uninterpreted event and associated recovery procedure
Several Matches	*General* 7	-> change recognized expected condition criticality -> modify recognized expected condition
	If two identical expected conditions at two different hierarchical level 8	-> remove one expected condition if expected twice -> modify one of them Advice to remove a goal: keep the high level goal if the user wants to be able to skip all the lower level procedures when the high level goal is satisfied; or keep the low level goal if the user only wants to skip the sub-procedure with the low level goal

Table 1: Failure types - preferred repairs Table

3.4. Refining the Heuristics for Preferred Repairs

Heuristic guidance, described in the previous section, assists the user in diagnosing a failure and choosing a repair. However, it is based on knowledge elicited from the knowledge engineer and thus may be incomplete. Moreover, the diagnosis stage corresponds to the complex process of explaining failure in the domain in terms of the problem-solving components. Thus, heuristic guidance may also lack some domain specific knowledge, difficult to acquire without actually running the dynamic system. In order to solve those problems, we provide an extra level of automation which improves and refines this heuristic guidance over time with user feedback.

The preferred repairs associated to a given type of failure will be reinforced when actually selected by the user, or extended when the user selects another repair. The reinforcement increases the weight of a link which associates preferred repairs with a failure type. Contextual domain knowledge including for example, the particular task, user profile, type of sensors used and their reliability, and the active procedures, can be also be acquired for each link (as in Boy ,1990).

In our example, we suppose that the user has performed the suggested repair, selected the new expected condition and resumed execution. Later on in system operation, the user finds that once again, when the force-torque sensor was not working, the control assistant selected the "obstacle" abnormal condition. This failure now corresponds to case (7), the control assistant recognized both conditions "obstacle" and "force-torque sensor breakdown," and selected the most critical: "obstacle". In this case, the procedure acquisition assistant suggests modification of the criticality order between those conditions. Thus, the user notes that the heuristic guidance for the previous failure was incomplete, and that the criticality order of the

two abnormal conditions needed to be modified as well The user's feedback on the repair choice is used to modify and reinforce the corrected heuristic.

4. AUTOMATING THE REPAIR STAGE

After describing the various levels of assistance available to the user during the diagnosis stage of the procedure refinement task, we now present the automated KA tools useful for the repair stage. Once the user has chosen a repair, she has the choice between directly editing procedures or being guided further by the procedure acquisition assistant through the planning and performance of the repair. We first describe the lowest level of assistance available, the editors, then the repair guidance.

4.1. Performing the Repair

Basic capabilities for editing procedures have been implemented using structured editors. They include the Procedure, Condition, Elementary Condition and Elementary Action Editors (Figure 8). Those editors are directly accessible through the main menu. Editors are *structured* based on the chosen procedure representation (block representation) and integrate a number of syntactic constraints. For example, the user's selection of sub-procedures to be added to a procedure is

Figure 8: Procedure Editor

restricted to the list of existing procedures at a lower hierarchical level than the level of the edited procedure.The Elementary Action editor has been designed to make the model of the domain more transparent to the user. In order to create or modify an elementary action, the user directly "executes" this action on the user interface for control, by selecting a button or setting some numerical parameters for example. The procedure acquisition assistant automatically displays the expression of this action into the Action editor. This is possible since procedures are expressed as user's actions on the interface. For the Elementary Condition editor, all the objects in the domain model are accessible through selection lists to the user.

> In our example, the user decides to use the editors to add an abnormal condition describing the sensor breakdown event. She must first access the Procedure editor and display the "Approach Cube" procedure, then add the abnormal condition. The Procedure editor will display the list of existing abnormal conditions she may choose from. If the "sensor breakdown" condition doesn't exist, she must access the Condition Editor, and eventually the Elementary condition editor. Then she goes back to the Procedure editor and add the newly created abnormal condition in the ordered list of abnormal conditions of the "Approach Cube" procedure. She may then associate a recovery procedure with this new condition. If the procedure for approaching a cube without collision detection doesn't exist, she must first create it from scratch, then link it to the "sensor breakdown" condition in the "Approach Cube" procedure. This completes the repair.

Those editing capabilities should be sufficient for the knowledge engineer who knows which editing actions are needed for a given repair and in which order they should be performed. However they do not provide assistance for planning the repair to the domain expert or the operator, who are unfamiliar with the underlying knowledge representation.

4.2. Planning for the Repair

In order to provide more assistance to the user for planning and performing the chosen repair, PARC includes a procedure acquisition dialog which guides the user through the repair stage. It uses knowledge on how to translate a repair into a sequence of editing actions. It also takes advantage of the current execution state to narrow the set of options proposed to the user. Contrary to the heuristic guidance used during the diagnosis stage, this knowledge is mostly procedural and consists of more or less complex repair procedures associated to each possible repair (Mathé, 1991). These repair procedures do not depend on the domain, but only on the knowledge representation, its inference mechanism, and the current set of active procedures. Once the user chooses a repair, the appropriate repair procedure is selected and drives the dialog with the user. During this dialog, the user may be asked to describe how external conditions have changed, to specify new procedures, or to state which recovery procedure to associate with an abnormal condition and which active procedures and goals to drop. At some point during this dialog, the procedure acquisition assistant will also choose and display the appropriate editor to the user, where some slots have already been automatically filled in (deduced from the dialog with the user).

Once the repair has been completed or when the dialog is over, the Situation Description panel is updated to reflect the new state of the procedure base. The user may then select the appropriate expected condition and continue the execution. It is important to notice that the user doesn't have to perform a complete repair and may exit the dialog whenever she wants. Since the method is incremental, another related problem will be detected later on during the execution, or again in the same situation. Thus the user can choose how much to modify the procedure base each time she disagrees with the control assistant.

> In our example, the user chooses to add an expected condition describing the sensor breakdown event. The following dialog guides her through the repair:
>
> (1) PARC first asks the user to select a condition from the list of existing conditions or to edit a new one. If the "sensor breakdown" condition doesn't exist, PARC displays the Condition editor to the user.

(2) Given the current state of execution, PARC asks the user whether she wants to add this new condition as a goal or an abnormal condition. The user chooses to add an abnormal condition.

(3) PARC asks the user to select from the list of active procedures the procedure to which "sensor breakdown" will be added. The user selects the "Approach Cube" procedure. Then PARC display the list of existing abnormal conditions for this procedure and ask the user to insert the new condition in this ordered list. She inserts it after "obstacle".

(4) The procedure acquisition assistant finally asks the user to select an associated recovery procedure from the list of existing procedures at the same or higher hierarchical level, or to create a new one. The user chooses to create a new recovery procedure.

(5) To define this recovery procedure, PARC first asks the user to specify if the goal "gripper at position above cube" of the "Approach Cube" procedure may still be reached by a local recovery procedure when the force-torque sensor is down. If the goal is still valid, PARC also asks the user if the recovery procedure is an alternative procedure or an intermediate procedure (to be executed before going back to the current procedure "Approach Cube"). The user answers that an alternative procedure is needed (Figure 9).

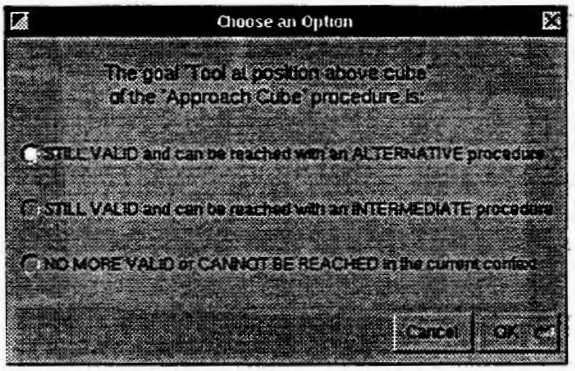


Figure 9: Part of the Repair dialog

(6) From this information, PARC automatically deduces that the new alternative procedure's goal is "gripper at position above cube," pointing towards the same next procedure "Align Gripper," and belonging to the same context "Pick up Cube". It displays this information in the Procedure editor, and the user only needs to enter the new procedure name and actions needed for approaching a cube with no force-torque sensor.

When the repair is done, the procedure base is updated (Figure 10), and the Situation Description panel graphically displays the new expectation "sensor breakdown" attached to the "Approach Cube" procedure and its associated new recovery procedure. The user may then choose this condition and continue the execution.

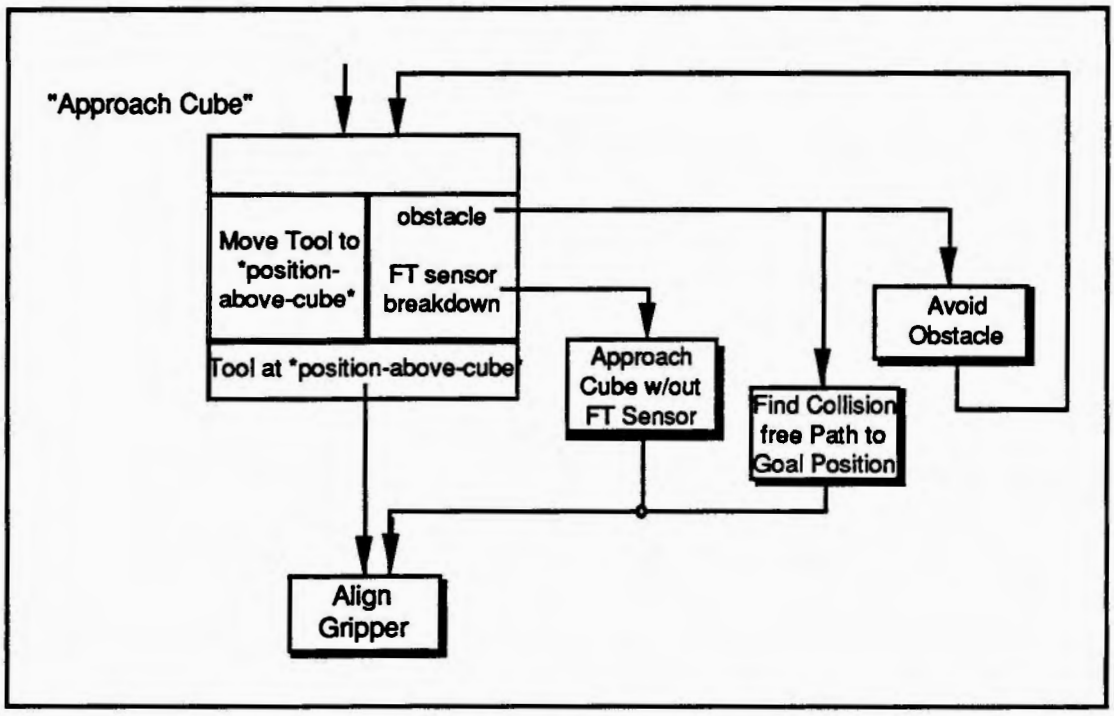


Figure 10: Part of "Pick up Cube" procedure after repair

5. DISCUSSION

In Sections 3 and 4 we presented the procedure acquisition assistant guiding the user in the diagnosis and repair of procedures. We demonstrated that PARC is able to assist the user in all four steps of the procedure refinement task by providing increased levels of assistance. The modules of the control assistant and of PARC are at various stages of completion. The control assistant, which includes a friendly user interface, is currently operational. It is able to control the Puma 560 robot arm for the robot assembly task. The structured editors and display of current execution state in PARC are implemented. The procedure acquisition dialog, and the heuristic guidance for diagnosis and repair, are designed and currently being implemented. We are currently in the process of completing the design for the reinforcement of heuristics from user feedback. Modules are implemented in C, Objective C, Common Lisp on Sun and NeXT workstations, and communicate through the Task Control Architecture (Fedor & Simmons, 1991).

We now analyze PARC in terms of the taxonomy for interactive knowledge acquisition tools proposed by Musen (1989). Instead of placing PARC at a fixed level in the taxonomy, PARC can be viewed as moving through various levels of the taxonomy with increased assistance. At the least level of assistance, the structured editors in PARC require knowledge of the domain, the problem-solving method, and the knowledge representation. Thus, at this level PARC can be seen as a "method level" tool, useful for knowledge engineers. However, the heuristic guidance assists the user in diagnosing and repairing a failure - automating the process of expressing domain knowledge in terms of components of the problem-solving method. Thus, this increased assistance makes the knowledge level analysis more transparent to the user. Finally, PARC already includes domain specific knowledge on the task and the robotics domain. Moreover, the refinement of the heuristic guidance with user feedback can be seen as tailoring the guidance to the user and the task by acquiring domain dependent knowledge, making PARC a "task level" tool, potentially more usable to domain experts.

Another important aspect of PARC concerns its representation for procedures, which can be seen an an intermediate representation for communicating with users in reactive control domains. The

block representation narrows the gap in representation mismatch (Representation mismatch (Buchanan *et al.*, 1983) refers to the gap between the user's knowledge, and the knowledge needed in order to communicate with the KA system.) In PARC, most steps in procedure acquisition require that the user be familiar with the reactive control method and the procedure representation. Does that present the typical user with too great of a representation gap? This can only be answered definitively through experimentation with PARC. However, we have reason to believe that our reactive control method and block representation do not present much of a gap because they were developed as appropriate cognitive representations for operators of dynamic control systems, based on the following:

(1) We designed this model following a cognitive analysis of the robotic control task;

(2) This model is confirmed by studies in other domains involving a reactive control task;

(3) In reactive control domains, operational knowledge is represented in manuals using terms such as "procedures," "nominal," "emergency situations," "recovery procedures," etc., whose semantics are well known to the domain experts (both system designers and operators);

(4) From our experience on two different robotics tasks, the elements of the reactive control method seem to be familiar and intuitive to the domain experts.

6. RELATED WORK

Knowledge acquisition can be broadly classified into three stages: *elicitation*, *refinement* and *restructuring* (Bareiss *et al.*, 1989). Currently we have automated only the *refinement* stage of the procedure acquisition process. The *elicitation* and *reformulation* stages are done manually. Related work on procedure elicitation automation has focused on providing graphical tools to the domain expert to assist her in encoding complex procedures with a visual language (Puerta *et al*, 1991; Saito *et al.*, 1991). Those graphical tools are very useful, but they need to be extended to facilitate the visualization of complex procedures during execution and repair. In particular, those approaches are based on flow diagrams. We believe that a more specific visual language is needed to represent procedures for reactive control. Most work on the refinement stage has been done for rule-based expert systems performing heuristic classification (Davis, 1977; Araki *et al.*, 1991). Some more recent work tries to automate the refinement process by including explanation-based learning technique (Pazzani *et al.*, 1991). Gruber (1989) proposed a method for refining strategic knowledge during reactive planning. This method is based on the induction of rules from examples, and is different from our method based on the block representation.

Reformulation has received little attention. In PARC, reformulation would enable the user to modify the structure of the procedure base into contexts of use. Such a context structure elicitation is actually done manually by interviewing the system designer. Grant (1991) has shown that the context structure for complex control tasks is incrementally built during system operation. In PARC, context restructuring is currently left to the user through the structured editors. However, Boy (1989) has demonstrated that the block representation could support the automation of such a restructuring process (Boy, 1989).

There have been recent attempts to integrate machine learning and knowledge acquisition methods (Wilkins, 1991). In our investigation, we integrated some ideas from machine learning for automated refinement of plans in reactive systems (Kedar *et al.*,1991) with our method. Most other machine learning efforts has investigated the problem of knowledge base and procedure refinement independently, under a number of guises. The approaches to refinement of knowledge bases in expert systems (e.g. Politakis & Weiss, 1984) typically perform induction over cases. Purely inductive approaches to refining procedures for dynamic systems include reinforcement learning (Lin, 1990) and active experimentation (Gil, 1991; Christiansen *et al.*, 1990). Machine learning approaches that are not purely inductive (i.e. using a preexisting knowledge base) fall into four broad categories: (i) explanation-based learning from failure, which uses a complete/correct

knowledge base is to refine the incomplete/incorrect knowledge base (e.g. Hammond, 1986; Chien, 1989); (ii) induction and active experimentation when the explanation is insufficient (e.g. Rajamoney & DeJong, 1988; Pazzani *et al.*, 1991; Ourston & Mooney, 1990); (iii) acquisition through apprenticeship learning from the user (e.g. Laird *et al.*, 1990; Wilkins, 1988) and (iv) using one approximate knowledge base to refine another (e.g. Kedar *et al.*, 1991).

7. FUTURE WORK AND CONCLUSION

In order to evaluate PARC, a set of experiments will be conducted this year with users in the robotics domain. The following hypotheses are among those we plan to test: (1) Does PARC increase the accuracy and/or efficiency of procedure refinement (over manual refinement, in comparison with other procedure refinement tools)? (2) Does each level of assistance provide increased accuracy and/or efficiency? (3) Is the procedure representation appropriate for the typical users of this system? In order to perform these experiments, a rigorous measure of 'accuracy' and 'efficiency' of the procedure base needs to be defined and measured (using the number of execution failures, aborted goals, execution time, repeated failures, etc.). We also need to define metrics of comparison among different refinement methods.

While our procedure acquisition method is not dependent on the specifics of the robotics domain, we are evaluating its reusability by exploring its application to the Space Shuttle Mission Control procedures. Our method currently addresses only some of the issues in supporting procedure management and maintenance for complex systems. In particular, this methodology assumes that procedure refinement is done as an individual task, and that time is available to interact with PARC to correct the procedure base. This makes it more suited to development of procedures on simulator by small teams. More work needs to be done to study how procedures are designed and refined by corporations such as airlines companies, where operators are only remotely involved in the process (Degani *et al.*, 1991). Furthermore, tools for evaluating the consequences of choosing one procedure over another or of modifying a procedure could be very helpful during the refinement process.

We are at the preliminary stages of implementing the heuristic guidance. The solution proposed only acquires new links or reinforces/weakens existing links between failures types and repairs, but it cannot currently acquire new failures types. Only some of the failures types could be acquired from the user. In particular, those are failure types that can be discerned directly from the graphical representation of the current execution state (e.g. a missing link between an abnormal condition and a recovery procedure for it). We are currently considering a technique similar to the one proposed in (Gruber, 1989) for acquiring rules associating new failures with repairs by induction from examples of failure types and repairs.

We plan to improve the flexibility of PARC. Currently, the levels of assistance are hard-coded in PARC. PARC could be made more modular, letting the user select the levels of automation during the procedure refinement process. A user-profile could also be used to choose a default level of assistance for different types of users.

We also plan to enhance the user interface for control by adding several features: higher level sensor predicates more useful to the user, and a 3D simulator (with special glasses) in order to simulate a new procedure before executing it with the real robot.

Finally, we plan to develop a graphical visual language for the procedure representation, assisting the user in exploring the procedures base, displaying an execution trace in the procedures network, and letting the user edit procedures graphically. Those graphical tools, along with a good indexing scheme and query language, should help in scaling up our method to larger and more complex procedure bases.

Acknowledgements

Many thanks to Guy Boy and Lonnie Chrisman for their insightful comments on this project. Thanks to Jay Steele for developing the robotics environment and to Jonathan Kolyer for developing the user interface on the NeXT. Thanks to Reid Simmons and Chris Fedor of Carnegie-Mellon University for providing and supporting the Task Control Architecture. Thanks to Guy Boy, Peter Friedland, Kate McKusick (NASA Ames) and Brian Slator (ILS) for reviewing this paper.

References

Araki, D., Kojima, S. (1991). KASE Project towards Effective Diagnosis System Developments, Proceedings of the 6th Knowledge Acquisition for Knowledge-Based Systems Workshop, Vol 1, Banff, Canada, October 6-11.

Bareiss, R., Porter, B.W., Murray, K.S. (1989). Supporting Start-to-Finish Development of Knowledge Bases, *Machine Learning*, 4(3-4):259-283.

Boy, G.A. (1989). The Block Representation in Knowledge Acquisition for Computer Integrated Documentation, Proceedings of the 4th Workshop on Knowledge Acquisition for Knowledge-Based Systems, Banff, Canada, Oct. 1-6.

Boy, G.A. (1990). Acquiring and Refining Indices According to Context, Proceedings of the 5th Workshop on Knowledge Acquisition for Knowledge-Based Systems, Banff, Canada, Nov. 4-9.

Buchanan, B. G., Barstow, D. K., Bechtel, R., Bennett, J., Clancey, W., Kulikowski, C., Mitchell, T., and Waterman, D. A., (1983). Constructing an Expert System, in Hayes-Rogh, F.,Waterman, and D. A., Lenat, D. B. (Eds.), *Building Expert Systems*, Reading, MA, Addison-Wesley.

Chien, S. A. (1989). Using and Refining Simplifications: Explanation-Based Learning of Plans in Intractable Domains, Proceedings of the 11th IJCAI Conference, Detroit, MI.

Christiansen, A. D., Mason, M. T., and Mitchell, T. M. (1990). Learning Reliable Manipulation Strategies without Initial Physical Models, Proceedings of the IEEE International Conference on Robotics and Automation.

Davis, R. (1979). Interactive transfer of expertise: Acquisition of new inference rules, *Artificial Intelligence*, 12(2):121-157.

Degani, A., Wiener, E. L. (1991). Philosophy, Policies, and Procedures: The Three P's of Flight-Deck Operations, Proceedings of the 6th Int. Symposium on Aviation Psychology, Columbus, Ohio, April 29- May 2.

Fedor, C., & Simmons, R. (1991). Task Control Architecture User Manual, Manual Version 5.2, Carnegie-Mellon University, June 1991.

Gil, Y. (1991). A Domain-Independent Framework for Effective Experimentation in Planning, Proceedings of the Eight Workshop on Machine Learning, Evanston, IL, June 27-29.

Grant, S.A. (1991). Modelling Cognitive Aspects of Complex Control Tasks, Ph.D. Dissertation, 1991, University of Strathclyde, UK.

Gruber, T.R. (1989). Automated Knowledge Acquisition for Strategic Knowledge, *Machine Learning*, 4(3-4):293-336.

Hammond, K. J. (1986). Learning to Anticipate and Avoid Planning Problems Through the Explanation of Failures, Proceedings of the Fifth AAAI Conference, Philadelphia, PA.

Kedar, S., Bresina, J., Dent, L. (1991). The Blind Leading the Blind: Mutual Refinement of Approximate Theories, Proceedings of the Eigth Workshop on Machine Learning, Evanston, IL, June 27-29.

Laird, J. E., Hucka, M., Yager, E. S., and Tuck, C. M. (1990). Correcting and Extending Domain Knowledge Using Outside Guidance, Proceeding of the Seventh International Machine Learning Conference, Austin, TX.

Lin, L. (1990). Self-Improving Reactive Agents: Case Studies in Reinforcement Learning Frameworks, Proceedings of the International Conference on Simulation of Adaptive Behavior, Paris, France.

Mathé, N. (1991). Second Progress Report on "Procedure Management and Maintenance", European Space Agency internal report, August 31.

Mathé, N. (1990a). A Blackboard Approach to Intelligent Assistance for Space Telemanipulation, AAAI-90 Workshop on Blackboard Systems, Boston, Massachusetts, July 29.

Mathé, N. (1990b). A Space Remote Control Application : Cognitive Modeling and Blackbaord-Based Implementation, Proceedings of the 3rd International Conference on Human-Machine Interaction and Artificial Intelligence in Aeronautics and Space, Toulouse, France, Sept. 26-28, pp 377-392.

Mathé, N. (1990c). Intelligent Assistance for Process Control: Application to Space Teleoperation, Thesis Dissertation (in french), Nov. 90, ENSAE, Toulouse, France.

Musen, M. A. (1989). Conceptual Models of Interactive Knowledge Acquisition Tools, *Knowledge Acquisition*, 1(1):73-88.

Musen, M. A., Fagan, L.M., Combs, D.M., & Shortliffe, E.H. (1987). Use of a domain model to drive an interactive knowledge editing tool, *International Journal of Man-Machime Studies*, 26(1):105-121.

Nielsen, M., Grue, K., Lecouat, F. (1991). Expert Operator's Associate: a Knowledge-Based System for Spacecraft Control, Proceedings of the 1991 Goddard Conference on Space Applications of AI, Greenbelt, Maryland, May 1991.

Ourston, D., Mooney, R.J. (1990). Changing the Rules: A Comprehensive Approach to Theory Refinement, Proceedings of the 8th AAAI Conference, Boston, MA, pp 815-820.

Pazzani, M.J., Brunk, C.A. (1991). Detecting and correcting errors in rule-based expert systems: an integration of empirical and explanation-based learning, *Knowledge Acquisition*, 3:157-173.

Politakis, P. and Weiss, S. M. (1984). Using Empirical Analysis to Refine Expert System Knowledge Bases, *Artificial Intelligence* , 22(1):23-48.

Puerta, A., Egar, J., Tu, S., Musen, M. (1991). A Multiple-Method Knowledge-Acquisition Shell for Automatic Generation of Knowledge-Acquisition Tools, Proceedings of the 6th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, Vol 2, Banff, Canada, October 6-11.

Rajamoney, S. A. and DeJong, G. F. (1988). Active Explanation Reduction: An Approach to the Multiple Explanations Problem, Proceedings of the Fifth Machine Learning Conference, Ann Arbor, MI.

Saito, T., Ortiz, C., Loftin, R.B. (1991). On the Acquisition and Representation of Procedural Knowledge, Proceedings of the 6th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, Vol 2, Banff, Canada, October 6-11.

Wilkins, D. C. (1988). Knowledge Base Refinement Using Apprenticeship Learning Techniques, Proceedings of the Ninth AAAI Conference, St. Paul, MN.

Wilkins, D.C. (1991). A Framework for Integration of Machine Learning and Knowledge Acquisition Techniques, Proceedings of the 6th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, Vol 2, Banff, Canada, October 6-11.

Using Causality as a Generic Knowledge Representation, or Why and How Centralised Knowledge Servers Can Use Causality

Tim Menzies, Ashesh Mahidadia, Paul Compton

Artificial Intelligence Lab, School of Computer Science and Engineering,
University of New South Wales, P.O. Box 1, Kensington, NSW, Australia, 2033.
{timm|ashesh|compton}@spectrum.cs.unsw.oz.au.

Abstract: A growing consensus in the knowledge acquisition community is that knowledge bases (KBs) do not contain some absolute truth but are merely a currently satisfying model of the world which will undergo continual refinement and evolution. KBs are a construct dependant on the context of their construction. Tools for understanding and debugging context-dependant constructs should be intrinsic to the representation, The need for such support is crucial for centralised knowledge servers that will need to arbitrate between divergent knowledge constructed from different sources. Good explanations of internal processing can quickly identify faulty models and, thus, prompt the evolution of the knowledge base. Rather than bolt on a post-hoc explanation facility to such a system, we advocate a generic knowledge environments based on explanation (or *causal*) links between entities in the domain.

1. INTRODUCTION

"One ring to rule them all."
J.R.Tolkein

Consider the task of creating some centralised repository of knowledge that is to be shared and updated by a diverse population. How should we structure the representations in this repository such that these representations support the knowledge acquisition (KA) process? This question is central to our research. *QMOD* is a *hypothesis testing environment* with a long term goal of allowing a community of scholars to contribute models and data relating to their current theories to a centralised knowledge base (Feldman, 1989a; Feldman, 1989b; Menzies, 1992a; Menzies, 1992b). Our initial test domain is neuroendocrinology and our current tools are aimed at a single researcher. We dream of the day when researchers around the world record their models and data into some form of electronic book that "wakes up" and complains when a researcher tries to record something that disagrees with what is already in the system.

A generalisation of *QMOD* is *CHuTE*[1], a *customisable* hypothesis testing environment. *CHuTE*'s aim is to allow the combined processing of different kinds of knowledge. Our current list of knowledge types relevant in our test domain is propositional knowledge; quantitative equations; qualitative statements; and explicit causal statements. *QMOD* dealt with compartmental models, a structure representing causality and conservation of mass, but traditionally modelled using differential equations. We suspect that this list of knowledge types will grow with time. Ideally, we would like a system that could be easily extended to cover new types of knowledge. In order to manage such diversity, the *CHuTE* processing algorithms need to be organised around some underlying common features of these different representations.

---

1 Pronocunced "shoot" or "cute" depending on how it is typeset.

Agreement on the nature of generic knowledge representations shared by different KRs does not yet exist in the KA/KR literature. (Gaines 1990) observes that the field of KA methodologies, techniques, and tools is marked by a diverse set of (mostly) incompatible tools and techniques. KR methodologies, techniques, and tools are in a similar divergent state. We can identify at least three major approaches: (i) knowledge fragments (Lenat, 1990; Neches, 1991; Futrelle, 1992); (ii) task-centred architectures (Chandrasekaran, 1986; Steels, 1990; Marques, 1991; Tu, 1991; Patzer 1992); (iii) object-oriented representations (Brachman, 1985; Skuce, 1989; Gaines, 1990; Rappaport, 1991)); as well as at least one other emergent and potentially-influential KR technique: (iv) graph operations on runtime-generated symbolic models (Clancey, 1992)[2].

A tacit assumption in traditional knowledge engineering and, we suspect, some of the above approaches is the *Feigenbaum simplification:* i.e. KA is a process of generating a copy of a body of knowledge that currently exists in the minds of experts which, in turn, is an accurate reflection of the world (Feigenbaum 1983). We believe this simplification is improbable. "Knowledge" as stored in a KB is dependant on who says it, and even when it is elicited. (Shaw, 1988) took a group of geological experts and had them build knowledge bases for the same domain, review their own knowledge bases twelve weeks later, and review each other's knowledge bases. For the twelve week review study, it was found that an expert's understandability and agreement with their own knowledge was less than total. For the cross-expert review, it was found that experts disagree significantly with each other. (Compton, 90) documents an expert system which was developed using an iterative prototyping methodology. Such systems never reach a logical termination point. There may always be one more major insight into the domain, one more major conceptual error, and one more significant addition. (Feldman, 89a) and (Compton, 90) claim that our explicit knowledge representations are not records of structures inside the head of an expert. Rather, the "knowledge" tricked out of an expert is a report customised to the specific problem, the specific justification, the expert, and the audience; i.e. "truth" as expressed by human experts is a variable. More precisely:

> *Knowledge is a function of the context in which it is expressed.*

If we relax the Feigenbaum simplification, then we should always suspect that our knowledge bases may not reflect either the domain expert's internal model or the real world. Consequently, our central repository will be the focus of much debate as different experts argue about the accuracy of each other's contributions. We do not view this as a death-blow to the goal of building centralised knowledge servers. Rather, it means that our knowledge servers should be designed such that they support the argument process. KA is not a process of "mining the jewels in an expert's head" (Feigenbaum, 1983)[3]. (Compton, 1992a) argues that knowledge is a dynamic construct and that our KBs should be tools for exploring those constructs rather than tools for enshrining old insights. KA is a process of building *and debugging* a symbolic model that may never have been created previously. This view is endorsed by Bradshaw *et al* who argue that the primary utility of...

> *a model is not to serve as a "picture" of a domain, but as a device for the attainment or formulation of knowledge about it.* (Bradshaw, 1991, p4-2)

---

2 If the reader is dismayed to see that their favourite KR technique is not on our list, then we have made our point regarding the divergent nature of KR and the lack of arguement between practioners regarding the "best" KR technique.

3 The "jewels" we find there will be dependant on the shape of the drill we use to dig for them and the properties of the jeweller's glass that we elect to examine them with.

(Popper, 1963) argues that theories can never *proved* in some absolute sense. Instead, the currently "true" theories are merely the ones that have not been disproved. A corollary of this view of knowledge is that *all* "true" theories should be reviewed in the light of any new evidence since this new evidence could disprove the currently "true" theories. KR languages that do not assume the Feigenbaum simplification need to support active verification modules that check all new contributions to the knowledge base against existing knowledge and existing test data. An auto-verification module can significantly simplify KA. Ripple-down-rules is a rule language augmented by knowledge of the context in which the rules were elicited. This meta-knowledge allows the system to automatically propose fixes to incorrect classifications. This technique is the enabling technology for a knowledge acquisition shell that permits a domain expert to build and maintain large knowledge bases (> 1000 rules) with no assistance from a knowledge engineer (Compton, 1992b). The generic KR techniques mentioned above are silent on the issue of verification. For example, Futrelle and Patzer *et al* do not discuss how they will check the contents of their knowledge bases. We view this as an error. The *QMOD* experience has been that rigorous executable models of existing medical knowledge contain many inconsistencies (see below). We conjecture that this result applies to other technical literature (e.g. the ecological literature being studied by Patzer *et. al*). If we are correct, then KRs that assume the Feigenbaum simplification will be overwhelmed by inconsistencies.

CHuTE rejects the Feigenbaum simplification. Knowledge within *CHuTE* should be under constant review/maintenance. The essential functionality of *CHuTE* (and, we believe, all knowledge bases) is knowledge test-and-revision; i.e. exploration. Rather than bolt on a post-hoc explanation facility to our KR, *CHuTE* is implemented using an experimental explanation-based representation that utilises the causal links between entities within the knowledge base. *CHuTE* compiles its various input knowledge representations into an underlying directed graph. The arcs of these graphs are causal links between entities in the representation.

The rest of this paper describes our work in this area. Section Two describes our formalism while Section Three gives some experimental results. Section Four describes related work. The appendix contains code for one of the examples discussed in Section Three.

2. *CHuTE*

2.1. Basic Definitions

Defn. 1:	An "hypothesis" is knowledge that can be doubted.

We argued above that every knowledge base should be actively and continually doubted; i.e. all knowledge (and all knowledge bases) are hypotheses.

Defn. 2:	A "hypothesis testing environment" is a knowledge-based program that mistrusts its own knowledge.

By this we mean that along with the inference engine and knowledge base, the system also contains critics that assess the KB according to (e.g.) if the knowledge base can explain known observations. Such a system automates part of the Popper knowledge refinement cycle and supports the active debugging of symbolic models.

Defn. 3:	A "*customisable* hypothesis testing environment" (*CHuTE*) is a hypothesis tester that permits the extension of the types of knowledge that it can process.

CHuTE executes via an underlying causal model. We need to be able to convert from a statement in some KR to a causal model.

Defn. 4:	"K_i" is a knowledge-based system labelled i. Each K_i is a pair of: *<Assertions, InferenceEngine>*.
Defn. 5:	"$K_{i[j]}$" is an assertion uniquely identified by j within K_i .
Defn. 6:	"T_i" is a program that inputs an $K_{i[j]}$ and outputs contributions to M; i.e. it converts domain models to causal models.

2.2. Defining Causality

T_i assumes that $K_{i[j]}$ is shorthand for a set of causal links M. For example, within a forward-chaining knowledge base K_{fc}, we might read:

$K_{fc[1]}$ `if a or b or c then d`

This would contribute three elements to M:

$T(K_{fc[1]})$ = {d ← a, d ← b, d ← c}

The structure $A \leftarrow B$ can be read as "A is caused by B". But what do we mean by "cause"? Philosophically, the notion of "the cause" of an event is problematic. Bold statements such as A "caused" B (where A and B are some event) ignore extraneous events that may have also influenced B. Further, it is possible that B and A exist together merely accidentally and that we are confused into thinking that A causes B merely because of a statistically coincidence. We adopt a more pragmatic definition of causality that assumes the closed-world of a knowledge-based system.

Defn. 7:	"Causality" within knowledge-based systems are descriptions of the ordering of how an inference engine processed the variables within a model.

Note that our definition of causality is a combination of a model and an inference engine. Explanations of behaviour cannot be generated from acausal statements such as:

ashesh has legs = 2

Only when we combine this knowledge with knowledge of how this statement is processed (i.e. use the model *and* the inference engine) can we discuss causality.

We distinguish between domain models, qualitative models, causal models and causal pathways.

Defn. 8:	A "domain model" is a set of $K_{i[j]}$ statements that reflect a domain expert's attempt to model their domain, in all its complexity.
Defn. 9:	A "qualitative model" is an abstracted description of the domain model.
Defn. 10:	A "causal model" M is a directed graph whose nodes are measurable entities with internal state and whose arcs are indications of what nodes can be influenced by other nodes.
Defn. 11:	A "causal path" is record of an inference over a causal model. It comprises an ordered list of used links and the node states encountered/ set by the inference.

2.3. Converting Domain Models to Causal Models: Examples of *T*

2.3.1. $T_{propositional}$: Definitions for *T* for propositional systems can be found in (Simon, 1952). Simon's proposal is equivalent to the abstract model of propositional inference as defined by (Colomb 1990)[4]. (Colomb 90) recognises a special class of propositional rules called *B* rules. These are rules whose entire conditions are facts known prior to inferencing. Rules can be placed into a dependency graph and assigned a depth measure equal to the number of arcs between it and a member of *B*. (Simon, 1952) argues that conclusions in propositional systems are caused by conclusions at a lesser depth measure. For example, consider the following rules in our forward chaining language K_{fc}

```
kfc[2] if infant or moron then not_legally_responsible.
kfc[3] if age(N) and N > 7 then infant.
kfc[4] if not_legally_responsible and guilty then jail = false.
kfc[5] if motive and means and opportunity and witnesses then guilty.
```

Its associated causal model is shown in Figure One.

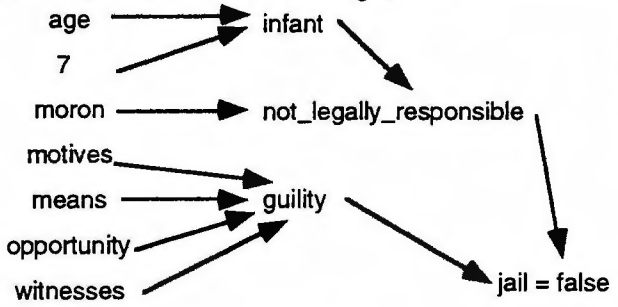


Figure One: Causality in a propositional system

2.3.2. $T_{equations}$: Consider the bathtub shown in Figure Two.

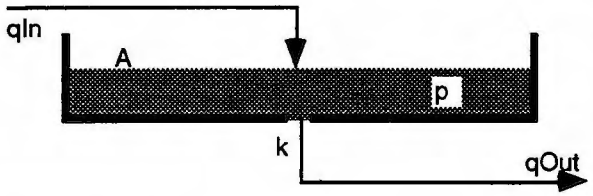


Figure Two: A bathtub (from Iwasaki, 1988a)

The domain model for this system could be represented in a equational representation: K_{eq}.

```
Keq[1]:    k = c3.
Keq[2]:    qOut = k * p.
Keq[3]:    a = c1 * p.
Keq[4]:    qOut = qIn.
Keq[5]:    qIn = c2.
```

qOut is the quantity of water flowing out of the system; *qIn* is the quantity of water flowing into the system; *p* is the pressure at the bottom of the bathtub; *a* is the amount of water in the bathtub; *k* is the valve opening at the base of the tub; and *c1*, *c2*, and *c3* are constants. In English, the equations are saying that (1) the value opening (*k*) and (5) the input flow rate (*qIn*) are exogenous (externally controlled); (2) the output rate (*qOut*) is proportional to the pressure (*p*); (3) the pressure (*p*) is proportional to the amount of water (*a*); and (4) when the system is in equilibrium, the input flow (*qIn*) equals the output flow (*qOut*).

---

[4] Note that Colomb's work was motivated as an optimisation technqiue, and not an attempt to model causality.

The $T_{causal\text{-}ordering}$ algorithm of (Iwasaki & Simon, 1986) first converts this domain model into a qualitative model that specifies that there exists certain inter-related variables:

```
Keq[1]':    [k,c3]
Keq[2]':    [qOut,k,p]
Keq[3]':    [a,c1,p]
Keq[4]':    [qOut,qIn]
Keq[5]':    [qIn,c2]
```

Next, knowledge of the inference engine of a equational solver is used to deduce descriptions of the ordering of how an inference engine processed the variables within a model. Iwasaki & Simon's simple equation solver looks for collections of N statements in the qualitative model with N unknowns. Since N equations with N unknowns can be solved algebraically, $T_{causal\text{-}ordering}$ declares that these N unknowns are "solved" and the process repeats till all the variables are solved. If variables at cycle N require variables solved for at cycle N *-1*, then the latter are "caused" by the former. For our above example, the first cycle finds $K_{eq[1]}$ and $K_{eq[5]}$. Once their variables are declared known (*k* and *qIn*), the next cycle finds $K_{eq[4]}$ which declares *qOut* solved. The next two cycles uses $K_{eq[2]}$ to solve for *p*, then $K_{eq[3]}$ to solve for *a*. The resulting causal model is shown in Figure Three.

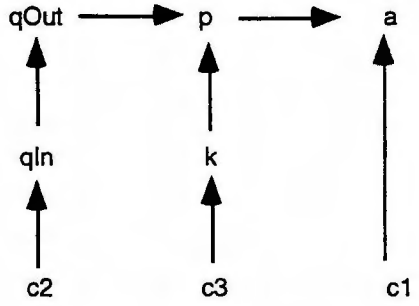


Figure Three:
A causal model for the bathtub.

In other $T_{equation}$ work, (Lee, 1992) uses heuristics to generate causal arcs that are "usually" acceptable to domain experts while (Ishida, 1989) and (Menzies, 1992c) use an analysis of the parse tree of the equations to generate their causal models.

2.4. Inverting *T*

We regard causal graphs as a hypothetical construct. We doubt that the causal models deduced by an analysis of a model and an inference engine *necessarily* reflects causality as understood by multiple experts that would use a centralised knowledge server. For example, the causal models deduced by causal ordering are crucially dependant on the choice of equations used to model the system. Since the choice of relevant equations is subjective and reflects the biases of the domain experts, then the resulting causal models are variable. (McIntosh & McIntosh, 80) describe six different mathematical models for human ovulation. At runtime, all the models exhibit the same behaviour, yet all are based on different assumptions.

The ability to revise causal graphs initialised and criticised by a variety of methods is the central concern of this research. Even if all the experts agreed on the nature of the causal model, causal ordering doesn't tell us how to handle a causal model that cannot explain known observations. Our commitment to the Popper paradigm implies we need to support proposed revisions to our causal models. However, there is a problem. If we critique the causal models and propose changes to them, then we need to be able to backtrack from these causal edits back to edits for the "home" KR. Hence the need for an inverse *T* function.

Defn. 12:	Let ΔM be a set of edits to M (our causal model). "$T^{-1}{}_i$" is a program that proposes edits to K_i such that it satisfies some critic.

For example, returning to $K_{fc[1]}$, suppose that our expert disagrees with $d \leftarrow a$. $T^{-1}{}_{fc}$ would propose a revision to $K_{fc[1]}$ such that it can no longer generate the offending causal link. That is:

```
                    M      = {d ← a, d ← b, d ← c}
                   ΔM      = retract d ← a
T⁻¹fc (M + ΔM)     = Kfc[1a] if b or c then d
```

Note that this revision process for retraction is more determinate than for assertions. In the case of:

```
ΔM      = retract d ← e
```

then $T^{-1}{}_{fc}\ (M + \Delta M)$ could be any one of a number of possibilities:

```
Kfc[1b] if a or b or c or e then d
Kfc[1c] if (a and e) or b or c then d
Kfc[1d] if a or (b and e) or c then d
Kfc[1e] if a or b or (c and e) then d
Kfc[1f] if (a or b or c) and e then d
Kfc[1g] if (a or b) and e or c then d
...
```

That if we compute $M_1 = T^{-1}(T(M_0))$, then perhaps M_1 will not equal M_0. When generating the domain model from the causal model, T^{-1} has less information available to it than T (since M is generated via qualitative model that contains less information than the original domain model). T^{-1} can only backtrack around its indeterminate points to propose variations to M. We can view this indeterminacy as a bug or a feature. As a bug we need to fix, we could perform partial matches between M_1 and M_0 to fill out the missing portions of M_1. As a feature, we could use T^{-1} as a generator of compromise models during some argument between a community of experts exploring a central knowledge server.

2.5. KA Using Causal Models

The above process allows us to provide insights into domain models via causal models. A causal model is generated via an analysis of a qualitative version of the domain model and its inference engine. Once generated, then explanations of behaviour, or failure to account for known behaviour, can be used to suggest revisions to the causal models. These insights can then be feed back into the domain model to prompt revisions/ refinements (see Figure Four). If a critic insists that a certain ΔM is required, then the system could reply something like "well, if you really mean that then that implies the following edit to the knowledge base". At which point the user replies "sure" or "oops, I guess I don't mean that at all".

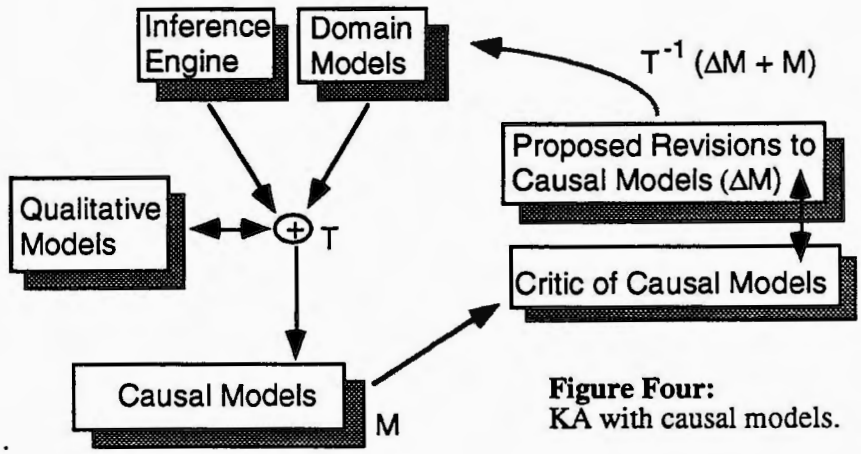


Figure Four:
KA with causal models.

3. EXPERIMENTAL WORK

This section discusses our research into critiquing a causal model. Our work to date has been focused on model critics that use databases of causes and known effects. This work was first presented to this conference as (Feldman, 1989a): an automatically generated critic of a large neuroendocrinological theory (Smythe, 1989). The critic was based on a difference analyser that reported effects not possible given the known causes and models. Prior to describing that complicated model, we present our standard tutorial example of our technique.

3.1. The Smythe '87 Study

(Smythe, 1987) proposes a inhibitory connection between serum corticosterone and neuro-noradrenergic activity (measured as the ratio of noradrenaline to its post-cursor, 3,4-dihydroxphenylethethyleneglycol (DHPG)). In order to verify this model, the (Smythe, 1987) study took a population of rats and measured their neuro-noradrenergic activity (*nna*), serum corticosterone (*cortico*) and adrenocorticotropin (*ACTH*) levels under various conditions:

- the *dex* group, which were given an injection of dexamethasone (*dex*) at 100 mg/kg;
- the *cold swim* group, which were subjected to a two minute swim in a bath of ice cold water;
- the *cold swim plus dex* group, which were subjected to both a cold swim and an injection of dexamethasone;
- the *control* group, which received no treatment at all.

Some of the collected data is shown in Table One.

Value	Control	Dex	Cold Swim	Dex + Cold Swim
nna (no units)	0.122	0.105	0.210	0.246
serum cortico (nmol/L	129	11.3	1232	32.8
serum ACTH (pg/ml)	89	0	240	0

Table One: Experimental Results from (Smythe, 1987).

The causal model under discussion is shown in Figure Five.

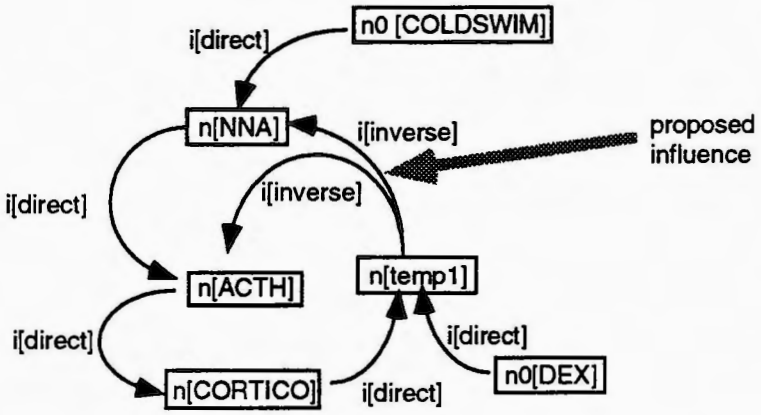


Figure Five: The (Smythe, 1987) model.

Figure Five shows a set of nodes representing continuos, real-valued variables connected by two types of influence arcs. After (Rajagopalan, 1986) we say that two nodes are *directly* linked (denoted *i[direct]*) if an upwards or downwards change in one pushes the other *up* or *down* respectively. Further, two nodes are *inversely* linked (denoted *i[inverse]*) if an upwards or downwards change in one pushes the other *down* or *up* respectively. Each node has a unique identifier and may have a measured or unknown value. In Figure Five, *n[temp1]* is unmeasured. This node is included for modelling purposes to indicate that *dex* and *cortico* have the same effects on the system. Certain nodes represent the exogeny to the system and are denoted as *n0*. These are boolean variables modelled as continuous variables whose values can be only zero or one. All the events that make up an experiment are represented as exogenous nodes. To represent (e.g.) *cold swim + dex*, we set *n0[DEX]* and *n0[COLDSWIM]* to 1. The control experiment is represented by setting all exogenous nodes to 0. When comparing two experiments, we compute the difference between the exogeny and set the deltas on the exogenous nodes accordingly.

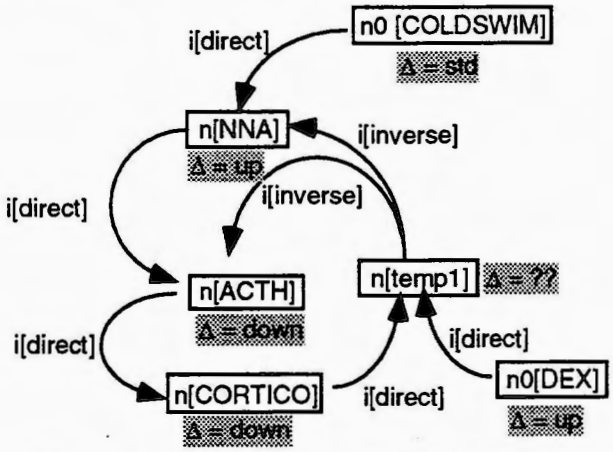


Figure Six: Changes from *coldSwim* to *coldSwim + dex*
(changes denoted with a Δ)

The changes to the state of the nodes between the *dex* experiment and the *dex + coldSwim* experiment are shown in Figure Six.

3.2. "Refutation via Explanation" vs "Validation via Simulation"

We can invalidate a model if we cannot find causal pathways from known effects (e.g. *acth down*) to known exogeny (*dex up*). If a pathway crosses a measured node, then the change of that node required for that pathway must be the same as the observed change. A pathway terminates if it arrives at an exogenous node and the change required for the pathway is the same as the known change in the exogenous variable. All the known changes are processed together; e.g. we search for causal pathways from *nna up* and *acth down* and *cortico down* to some exogeny. If a pathway needs to make some assumption about an unmeasured node (e.g. *temp1*), then this assumption must not contradict assumptions required for other pathways. We call this the principle of *no-conflicts-in-assumptions.*

This validation algorithm is a subtle optimisation method for the processing of qualitative models. Consider a qualitative system in which influences are propagated from exogeny (rather than *towards* exogeny as in our system). By definition, qualitative models are under-specified. A range of behaviours could result but note that we may only have measurements for a subset of the possible behaviours (see Figure Seven).

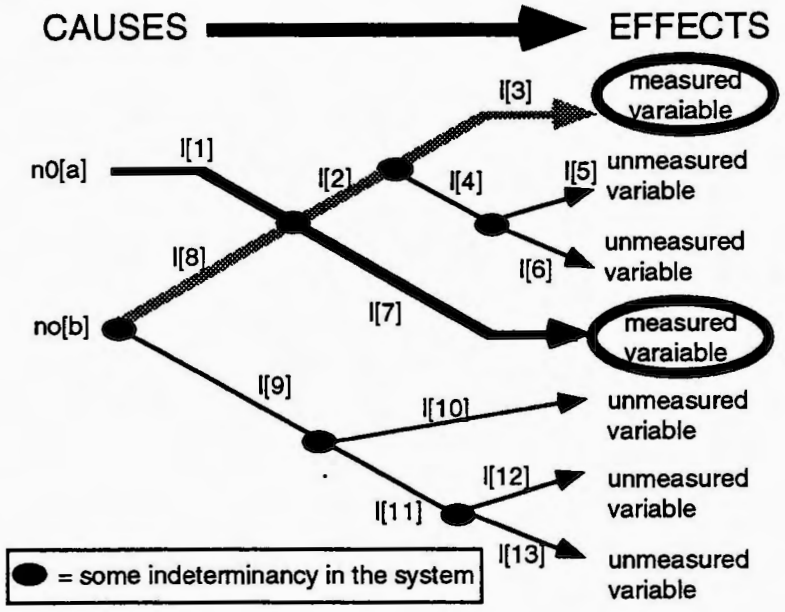


Figure Seven: Inferencing from causes to effects in a indeterminate qualitative model.

Consider how we would use such a program to try a Popper-style refutation of a model. We would generate all the behaviours, then check that all our causes and known effects are within the possible behaviours. Note that we would ignore the generated behaviours for which we have no expectation. The computational effort required to generate the behaviours for the unmeasured variables would hence be wasted.

Contrast this simulation-based approach with our algorithm. We constrain our search to the links from measured variables back to exogeny. Depending on the topology of the network, this may significantly reduce the number of links that need to be explored (compare Figure Seven to Figure Eight).

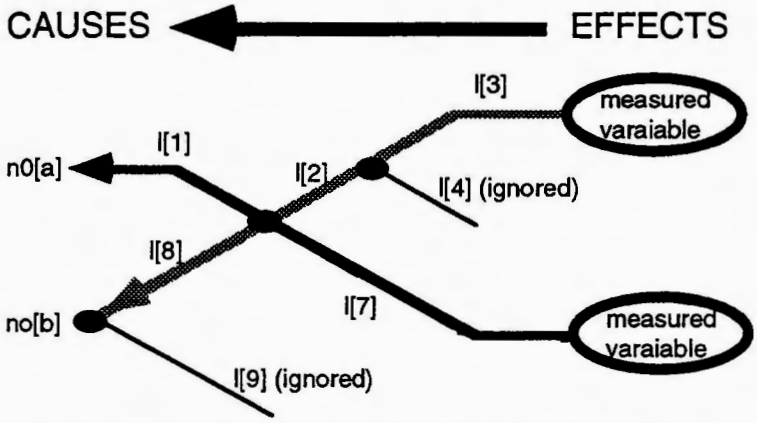


Figure Eight: Inferencing from effects to causes.

3.3. Indeterminacy , Causal Loops, and Qualitative Models

Figure Nine shows pathways that explain *cortico down* and *acth down*

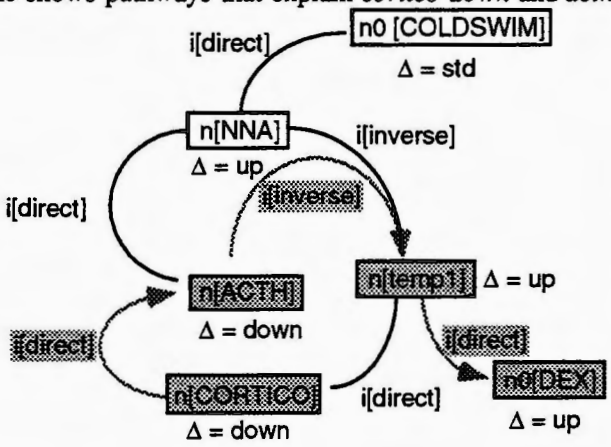


Figure Nine: Inferencing from effects to causes for (Smythe, 1987). Used pathways shaded.

Our explanation path assumes that *temp1* went up. Is this a valid assumption, given that *cortico* went *down* and Figure Five tells us that there is a direct link between *cortico* and *temp1*? The direction of change for *temp1* is surely indeterminate since two influences act on it *(a down* from *cortico* and an *up* from *dex*) and standard qualitative calculus is ambiguous regarding the addition of a qualitative positive and negative number.

In our system, we just make the assumptions necessary for the explanation. Our rationale is that we are interested in refuting models, not proving them. If a state change is possible, and it supports the current explanation, then we assume that state change. The reader may view this as too generous an assumption; that no causal model could be invalidated under these relaxed constraints. Fortunately, the principle of no-conflict-in-assumptions imposes enough constraints on our models such that we can still detect errors. For example, once the inferencing shown in Figure Nine assumes that *temp1* went *up*, then we suddenly cannot explain the observed change in *nna* (see Figure Ten).

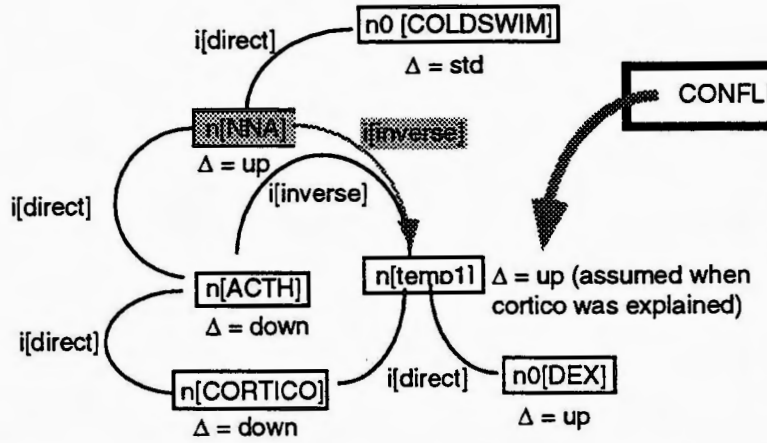


Figure Ten: Explanation path for *nna* going up fails at *temp1* since it must assume that temp1 Δ = *down*.

Note that our explanation-oriented algorithm can make definite statements about what behaviour (Smythe, 1987) cannot explain. The simulation-oriented algorithm described in Figure Seven would be unable to be so definite. Figure Eleven shows a propagation from causes to effects for (Smythe, 1987).

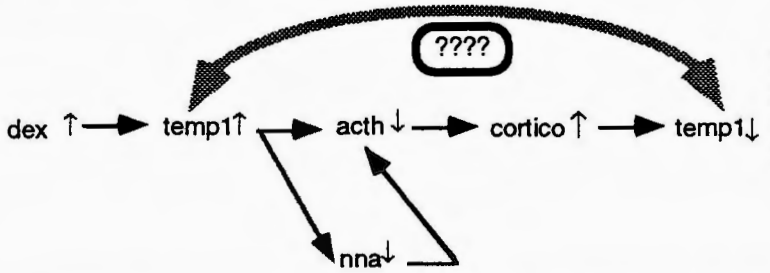


Figure Eleven: Inferencing from causes to effects for (Smythe, 1987).

The predicted changes to *acth*, *cortico*, and *nna* are dependant on the change in *temp1* but the feedback loop in the system results in a disagreement on its change. Propagating this disagreement permits the model to predict any change to the measured variables. Hence we conclude that the invalidation of qualitative models is best performed by explanation-oriented algorithms rather than simulation-oriented algorithms.

3.4. The Smythe '89 Study

(Feldman, 1989a) reports an analysis of the (Smythe, 1989) review paper and 343 data points collected from six related research papers. The summary model included 27 nodes, and 82 links. 109 of the data points could not be explained (32%). The types of inconsistencies included clerical errors in translating models into the representation. Some of the inconsistencies were due to deliberate simplifications of the model by the researcher. However, the most important result was that the norepinephrine data in hypothyroid rats who had been given an alpha-2 adrenegeric blocker could not be explained. This was a novel finding that the authors of the (Smythe, 1982) research paper were not aware of. Experiments are being conducted to investigate further certain hypothesis which this finding has prompted.

This result is interesting in that the authors of (Smythe, 1982) had only considered the effects of the alpha-2 adrenegeric blocker on hypothyroid rats rather than effect of hypothyroidism on alpha-2 adrenegeric blocker treated rats. That is, they had not considered the cross-experiment data comparisons. Although the data was highly statistically significant, the cross-comparison was not made since the authors were primarily interested in stress, not hypothyroidism. They therefore studied the effects of stress in the presence of hypothyroidism, to see whether or not

the same mechanisms were operative as in other stress situations. The reverse comparison looks at the effect of hypothyroidism in the presence of stress, a question that the authors were not addressing. The result is of importance since it suggests that the interaction between serotonin and norepinephrine described in (Smythe, 1982) will have to be relocated. This represents a major re-organisation of the (Smythe, 1989) model and to our understanding of the interaction between norepinephrine and serotonin.

3.5. Centralised Knowledge Servers and "Extra-Inferences"

These experiments with the neuroendocrinology literature suggests that published experimental data contains more information than the minimum required to verify the theory(s) for which it was collected. Using this extra-information, we can perform "extra-inferences"; i.e. make useful insights into theories that the theory authors have not made and using the data already existing in the publications. Note that the information required to gain our new insights into the Smythe models were available prior to this analysis but had not been noticed.

This result raises the fascinating possibility that within the existing scientific literature, there exists a backlog of extra-inferences awaiting the right tools to find them. Our existing "knowledge servers" (e.g. textbooks in libraries) should also be treated as hypotheses that should be rigorously analysed looking for inconsistencies.

3.6. Efficiency: Using "Breadcrumbs"

The causal models of many interesting domains include causal models with feedback loops (e.g. biology and process control). Give the presence of these loops, it is possible that the inferencing could take advantage of previous reasoning. For example, the explanation of *cortico up* in Figure Nine required an explanation for *acth down*. If our inference engine cached the nodes which it had explained and if *acth down* had been explained previously, then the search for the explanation to the change in *cortico* could terminate at *acth*.

As our algorithm walks over the net, it marks the trail that it is taking. We call this "dropping breadcrumbs". On failing to explain a particular result, it picks up the breadcrumbs on the failed path. Any breadcrumbs the search encounters that are not on the search's current path must have been dropped by previously successful searches. Consequently, we can terminate the search when we intersect with previously generated pathways; i.e. when we see breadcrumbs in front of us.

There are three kinds of breadcrumbs:

1) Measured effects that have been explained previously. This is the *Done* set.
2) Assumed changes. This is the *Assumes* set.
3) Records of measured effects that we tried to explain previously, but we could not. This is the *CantDo* set. Encountering this type of breadcrumb implies that this particular pathway is not productive. The search should avoid such bad paths and look for other options.

Our top-level loop runs over the set of measured effects (the *ToDo* set) and tries to explain each measurement. If an explanation is possible, it moves that measurement from *Todo* over to the *Done* set (successful searches). Any assumptions generated during the successful search are installed into the *Assumes* set. If the explanation fails, the measurement is moved from *ToDo* over to the *CantDo* set. While searching for an explanation, the program could arrive at another measured effect which has not been analysed before. In this case, the network traversal system is called recessively which may add further explained measurements or assumptions to the *Done* or *Assumes* set.

In order to demonstrate the algorithm, we extend the Figure Five model and explore the effects of earthquakes, big toes, and ear lobes on the body's reaction to stress (see Figure Twelve).

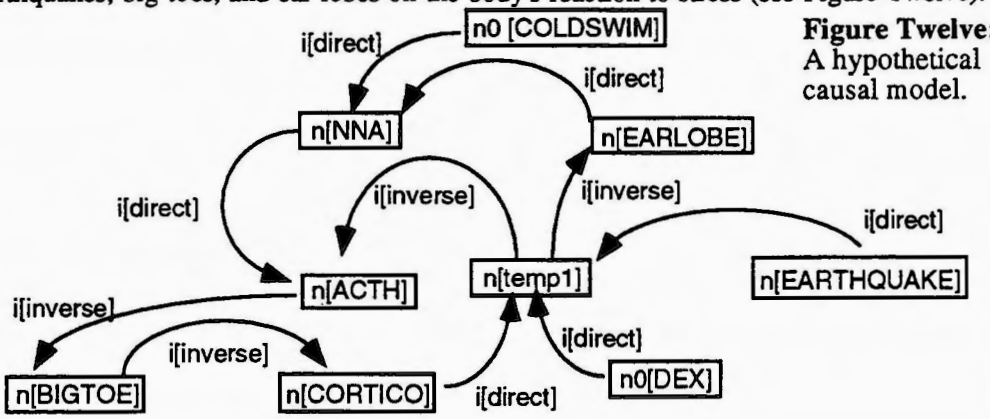


Figure Twelve: A hypothetical causal model.

Internally, this is represented by the following link assertions:

```
l(coldSwim,nna,direct).     l(nna,acth,direct).
l(acth,bigToe,inverse).     l(bigToe,cortico,inverse).
l(cortico,temp1,direct).    l(dex,temp1,direct).
l(earLobe,nna,direct).            l(temp1,earLobe,inverse).
l(temp1,acth,inverse).            l(earthQuake,temp1,direct).
```

We will assume that we are comparing two experiments:

```
E1 = [coldSwim,earthQuake],
E2 = [coldSwim,dex]
```

Assume that *nna* went *up*, *cortico* went *down*, and *acth* went *down*. Our program reports five traversals of the net[5].

```
?-  casual([nna(up), cortico(down), acth(down)], % measured effects
      [coldSwim(std), dex(up), earthQuake(down)], % known exogeny
      Explained, UnExplained, Assumptions)

Nº1   Explained   = [cortico(down), acth(down)],
      UnExplained = [nna(up)],
      Assumptions = [bigToe(up), temp1(up)]

[three deletions]

Nº5   Explained   = [nna(up)],
      UnExplained = [acth(down), cortico(down)],
      Assumptions = [temp1(down), earLobe(up)]
```

Either we can explain the *nna* change or the *acth* and *cortico* change. The crucial node is *temp1*. If the *nna* explanation arrives there first, then we assume that *temp1* has gone down. If the *acth/cortico* explanation arrives there first, we assume that *temp1* has gone up. Either assumption blocks the other explanation.

The efficiency of the breadcrumbs heuristic depends on the order in which the nodes were explored. The following trace of one fortuitous run shows how the system analysed the measured effects in the order: *[acth(down), cortico(down), nna(down)]*.

---

[5] The astute reader may have expected six traversals since there are six orders in which we can explore the set *[nna(up), cortico(down), acth(down)]*. However, our search heuristics cut out one result.

```
Analysing acth(down)
trying nna(down)                    % fails
trying templ(up)                    % propagates, leaving an assumption breadcrumb.
_ trying cortico(up)                % fails
_ trying dex(up)                    % propagates
_ _ dex(up) is a known cause        % found a n0
Q.E.D.                              % top-loop terminates leaving a successfully
                                    % explained breadcrumb

Analysing nna(up)
trying coldSwim(up)                 % fails
trying earLobe(up)                  % propagates, leaving an assumption breadcrumb
_ trying templ(down)
_ _ templ(down) fails due to assumption clash: templ(up)
                                    % breadcrumb clash, backtrack. Forget earLobe(up).
???                                 % top-loop terminates leaving an
                                    % unsuccessfully explained breadcrumb

Analysing cortico(down)
trying bigToe(up)                   % propagates, leaving an assumption breadcrumb
_ trying acth(down)                 % propagates
_ _ acth(down) was explained above  % breadcrumb found. search successful
Q.E.D.                              % top-loop terminates leaving a successfully
                                    % explained breadcrumb
```

The final assumption breadcrumbs are: *[bigToe(up), templ(up)]*. The code for this system is given in the Appendix.

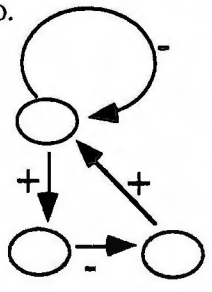
4. RELATED WORK

4.1. Qualitative Reasoning

This project's first attempt at domain modelling began with *QSIM* (Kuipers, 1986). Experience with that algorithm was the basis for our conclusion that it can be a waste of CPU to generate behaviours we can not criticise. Due to the indeterminacy of our models, *QSIM* generated more behaviours than we could assess. Hence, (Feldman, 89a) developed the current method. Our emphasis on explanation-oriented reasoning instead of simulation-oriented reasoning distances us somewhat from the standard qualitative reasoning approaches; i.e. confluences (DeKleer, 1984), qualitative process theory (Forbus, 1984) and *QSIM*. However, we note that we are closer to the qualitative mathematical approach of *QSIM* and the parts-centred ontology of confluences than the process-centred ontology of qualitative process theory.

Our work also has some connection with the phase-space portrait work of (Ishida, 1989). Rather than simulate a qualitative system to determine its time-dependant behaviour, Ishida uses results taken from process control theory to demonstrate that the eventual behaviour of certain causal models can be accurately predicted from an analysis of the eigenvalues of the system matrix. For example, the continuous values of the nodes in the model of Figure Thirteen will eventually converge to zero. We conjecture that in systems that exhibit massive feedback (such as our biological domain or Ishida's process control domain), that simulation-based qualitative methods are naive in their treatment of time.

Figure Thirteen: Eventually, feedback in this system will drive all the values of the nodes to zero.



4.2. Model Anomaly Localisation

(Darden, 1990) discusses theory anomaly localisation based on an analysis of the development of genetic theory in the early part of this century. While it was not their intention, the study also demonstrated the central role of causality in model anomaly localisation.

The technical appendix to the Darden study describes how their theory was represented in FR (a system based on a task-centred style of representation). While the FR representation was useful for structuring a complicated domain, most of the architecture was not needed for the anomaly localisation. The essential part of the implementation required for the localisation process were the causal links between parts of the theory (modelled as "function frames"). Anomaly localisation was a process of walking backwards from the final state back towards the initial state, inquiring at each point whether the intermediate state had been entered. Later versions of the program, as yet unpublished, are more sophisticated and used more of the FR architecture. Entities within the domain are bundled into groups (using functional knowledge) and anomaly localisation proceeds by groups, rather than by mere entities (Moberg, 1992).

The goal of the program used in the (Darden, 1990) study was to illustrate how a functional representation such as FR could yield a systematic generation of possible faults that could be fixed in a process of redesign. That is, unlike our work, they were exploring an existing representation rather than seeking the minimal architecture needed for model refutation. At most, we could argue that the (Darden, 1990) study demonstrates that in terms of model revision, the useful features of a representation are the causal links between entities. At the very least we observe that verification does not fall out straight away from a task-centred approach, but requires some work.

4.3. Diagnosis and Repair

(Struss, 1992) defines diagnosis as:

> *"a process of reasoning and acting in order to identify the cause of a malfunction in a way that allows the re-establish the desired functionality."*

In practice, this typically involves assigning state to variables connected together in some direct graph whose links denote how variables can effect each other. Diagnosis is a rapidly growing field (Hamscher, 1991), but most of the leading work assumes correct models (e.g. DeKleer, 1989). Characterising our work in terms of diagnosis, we could describe our system as being optimised for the detection of no possible diagnosis; i.e. recognising a faulty model. We use techniques from theory revision for model repair.

4.4. Theory Revision

QMOD is capable of demonstrating that an error in a theory exists, but it cannot identify and correct the error, leaving this task to the expert. Automatic methods of identifying and correcting errors in first order theories were first addressed formally by Shapiro in his *MIS* system (Shapiro, 1981). Techniques relating to theory revision, which extends theory to assimilate anomalous observations, are of specific interest in building and maintaining causal theory.

(Mahidadia, 1992a; Mahidadia; 1992b) describe a method for learning qualitative causal models, given experimental data. It takes experimental results to form a training set of positive examples. The program searches for qualitative causal relationships between identified concepts. Such qualitative causal relationships are found because they can explain experimental results. Together, these relationships will constitute the causal qualitative model that can explain all

experimental results. The program uses background knowledge in the form of qualitative causal relationships between the concepts.

4.5. Expert Critiquing System

Our approach has much in common with (Silverman, 1992)'s notion of an expert critiquing system (ECS) , which he defines as follows:

> *"programs that first cause their user to maximise the falsifiability of their statements and then proceed to check to see if errors exist. A good critic program doubts and traps its user into revealing his or her errors. It then attempts to help the user make the necessary repairs*[6]*".*

In the terminology of Silverman, our system has an deep model expert module (the causal network); a differential analyser based on exhaustive comparison; and no dialogue generator. It is unclear as to how to report the falsification of a theory in such a manner that it assists the user in fixing a faulty theory. For example: given that the change in *cortico* and *acth* could not be explained, but the only input arc into *cortico* is *acth*, then should we not omit the *cortico* error from the dialogue?

Silverman's research seems to be focused on an implementation-independent analysis of the process of "critiquing" a program. Our own research is for an engineering formalism that allows us to simultaneously process and validate different types of knowledge.

4.6. Belief Networks

(Pearl, 1991) describes techniques for deducing causal models (which he terms "belief networks") from observed frequency distributions of variables. His techniques make several assumptions that are not appropriate in our domain. Firstly, they are committed to the production of acyclic graphs. Biological systems often have feedback cycles. Secondly, they assume a large amount of data is available on all the entities in the domain. Our domain is characterised as being hypothesis-rich, but data-poor. Thirdly, once the causal models are initialised, the Pearl formalism does not permit review and restructuring of the deduced model. Such functionality is essential for our knowledge server. Fourthly, Pearl's approach does not permit the integration of an existing theory by an analysis of a new example. In our system, we can foresee that various users would treasure their favourite portions of the model (typically, the ones they have developed and successfully defended from all critics). It would be unacceptable to permit an algorithm to scribble all over this knowledge.

4.7. Software Verification

One way to characterise our work would be the application of results from the qualitative reasoning to the task of software verification. State-of-the-art verification tools are still very-much rule-based. Interestingly, they mostly based on and-or graphs of the rule base (Preece, 1992). We would call such graphs the causal model. Our work could be described as a general model-based verification technique for a wider class of knowledge-based systems.

4.8. Ripple-Down-Rules

In suggesting that understanding the inter-connections in a knowledge base is desirable for maintenance, we are not rejecting our earlier proposals for ripple-down-rules which hide the knowledge base structure and remove the need for knowledge engineers The practical utility of

---

6 ECSs are therefore much broader than the definition instantiated by *ATTENDING* (Miller, 1986) which had no mechanism for doubting its own internal knowledge base.

ripple down rules has already been proven, but only for maintaining classification systems (Srinivasan, 1991; Compton, 1991; Compton, 1992b). In the longer term we would hope to apply the ripple down rule strategy to more complex knowledge bases, such as those processed by *CHuTE* .

We note both our human ability to incorporate small chunks of new practical knowledge without difficulty and our ability to abstract from experience to make more intelligible models of the world. In our long term vision one should be able to simply add small chunks of immediately useable knowledge to knowledge bases in ripple down rule fashion, but as well, reason about this knowledge in more abstract ways. We have already noted some advantages in producing less brittle expert systems (Kang, 1991) by reasoning about the knowledge contained in a ripple down rule knowledge base.

Until we are able to properly synthesise these disparate approaches our research proceeds on the two separate fronts of maintenance which requires no understanding of the knowledge base and flexible manipulation and creation of new and complex knowledge structures based on generating causal understanding of any type of knowledge base.

5. CONCLUSION

"The concepts initially formed by abstracting from particular situations or experimental complexes acquire a life of their own."
Werner Heisenberg

The KB development process can be characterised as the building and debugging/evolving of symbolic models. We believe that in the case of centralised knowledge servers, a significant portion of the system's time will be spent in exploring criticisms of the knowledge in the server. Our experimental work indicates that within the neuro-endocrinological literature, there exist non-trivial numbers of extra-inferences awaiting the proper verification tools to find them . We suspect that this result generalises to the entire scientific literature.

Our readings of the generic representation literature suggests that current representations do not give verification sufficient primacy. Hence our proposal for a new, causal-based, representation. Causal explanations are a fundamental rhetorical device used in a multitude of domains. Their utility in discussing, debugging, and evolving a domain can not be over-emphasised. We believe that knowledge servers will require causal explanations frequently as multiple users struggle together to understand the intricacies of the KB. We suspect that causal explanations are a feature common to many KRs. As evidence of this, we note that techniques already exist for deducing causal models from equational and propositional systems.

Our QMOD/CHuTE research suggests that a generic verification engine can be built using causal models. We have demonstrated that the explanation-oriented processing of a qualitative causal model can make more definite conclusions regarding the impossibility of certain behaviours than a simulation-oriented system. Optimisation techniques for explanation-oriented refutation systems have been discussed. A general framework for KA using causal models has been proposed: i.e. a Popper-style refutation process is possible for all knowledge bases that support T-1 and T.

6. ACKNOWLEDGMENT

To our long-suffering domain expert, Dr. George Smythe of the Department of Chemical Pathology, St. Vincent's Hospital, Sydney, we acknowledge a large debt and marvel at his endless patience.

7. REFERENCES

Brachman R.J. & Schmolze J.G. (1985) *An Overview of the KL-ONE Knowledge Representation System* in **Cognitive Science** 9(2), April-June, pp171-216.

Bradshaw J.M, Ford K.M.,and Adams-Webber J.(1991) *Knowledge Representation of Knowledge Acquisition: A Three-Schemata Approach* in the **6th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop**, Banff, Canada, October 6-11, pp 4-1 to 4-25.

Chandrasekaran B. (1986) *Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design* in **IEEE Expert**, Fall, pp23-30.

Clancey W.J.(1992) *Model Construction Operators* in **Artificial Intelligence** 53,pp1-115.

Colomb R.M. & Chung C.Y.C. (1990) *Very Fast Decision Table Execution of Propositional Expert Systems* in **Proceedings of AAAI-90**, Boston Mass. USA, 30 July.

Compton, P. J. and Jansen, R. (1990). *A philosophical basis for knowledge acquisition.* **Knowledge Acquisition** 2: 241-257.

Compton, P., Edwards, G., Kang, B., Lazarus, L., Malor, R., Menzies, T., Preston, P., Srinivasan, A. and Sammut, C. (1991). *Ripple down rules: possibilities and limitations*, **6th Banff AAAI Knowledge Acquisition for Knowledge Based Systems Workshop**, Banff, 6.1-6.18

Compton, P. (1992a). *Insight and Knowledge* **AAAI Spring Symposium: Cognitive aspects of knowledge acquisition**, Stanford University, (in press).

Compton, P., Edwards G., Srinivasan A., Malor P., Preston P., Kang B., Lazarus L. (1992b) *Ripple-down-rules: Turning Knowledge Acquisition into Knowledge Maintenance* in **Artificial Intelligence in Medicine** (in press).

Darden L. (1990) *Diagnosing and Fixing Faults in Theories* pp319-353 in Sharager J. & Langley P. (eds) **Computational Models of Scientific Discovery and Theory Formation** Morgan Kaufmann Publishers Inc. San Mateo, California.

DeKleer J. & Brown J.S. (1984) *A Qualitative Physics Based on Confluences* in **Artificial Intelligence** 25, pp7-83.

DeKleer J. & Williams B.C. (1989) *Diagnosis with Behavioural Modes* in **Proceedings of IJCAI '89**, pp1324-1330.

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

Feldman B.T., Compton P.J., & Smythe G.A.(1989a) *Towards Hypothesis Testing: JUSTIN, Prototype System Using Justification in Context* in **Proceedings of the Joint Australian Conference on Artificial Intelligence**,pp319-331.

Feldman B.Z., Compton P.J. & Smythe G.A.(1989b) *Hypothesis Testing: an Appropriate Task for Knowledge-Based Systems* in Proceedings of the **4th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-based Systems Workshop**, Banff, Canada, October.

Forbus K. (1984) *Qualitative Process Theory* in **Artificial Intelligence** 24, pp85-168, 1984.

Futrelle R.P. (1992) *Machines Reading the Literature* in **Proceedings of the AAAI '92 Workshop on Communicating Scientific Knowledge**, San Jose, California, July 15.

Gaines B.R.(1990)*An Architecture for Integrated Knowledge Acquisition Systems* in the **5th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop**, Banff, Canada, November, 18 pages.

Hamscher W. (1991) *Principles of Diagnosis: Current Trends and a Report of the First International Workshop* in **AI Magazine** Winter, pp15-37.

Ishida Y. (1989) *Using Global Properties for Qualitative Reasoning: A Qualitative System Theory* in **Proceedings of IJCAI '89** pp1174-1179.

Iwasaki Y. & Simon H.A. (1986) *Causality in Device Behaviour* in **Artificial Intelligence** 29, 3-31.

Iwasaki Y.(1988), *Causal Ordering in a Mixed Structure* in **Proceedings of AAAI '88**, pp 313-318.

Kang B. & Compton P. (1991)*Towards a process memory*, **AAAI Symposium on Cognitive aspects of knowledge acquisition**, Boose J., Clancey W., Gaines B. && Rappaport A. (eds).

Kuipers B.(1986) *Qualitative Simulation* in **Artificial Intelligence** 29, pp229-338.

Lee, M., Compton, P. and Jansen, B. (1992). *Modelling with Context-Dependant Causality.* **Proceedings of JKAW'92**, Tokyo, in press

Lenat D.B.& Gutha R.V.(1990) *CYC: A Midterm Report* in **AI Magazine**, Fall, pp32-59.

Mahidadia A.J., Sammut C. & Compton P. (1992a), *Building and Maintaining Causal Theories*, in: **Proceedings of the AAAI Symposium on Knowledge Assimilation**, Stanford Spring Symposium, 1992.

Mahidadia A.J. , Compton P., Menzies T.J. , Sammut, C. Smythe G.A. (1992b) *Inventing Causal Qualitative Models: A Tool for Experimental Research*, in: **Proceedings of AI '92** , Hobart, Tasmania, November..

Marques D., Klinker G., Dallemange G.,Gautier P., McDermott J. & Tung D. (1991) *More Data on Usable and Reusable Programming Constructs* in **6th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop** Banff, Canada, October 6-11, pp 14-1 to 14-19.

McIntosh J.E.A. & McIntosh R.P. (1980) **Mathematical Modelling and Computers in Endocrinology** Springer-Verlag, Berlin.

Menzies T.J, Compton P., Feldman B. & Toft T.(1992a) *Qualitative Compartmental Modelling* in **Proceedings of the AAAI Symposium on Diagrammatic Reasoning**, Stanford, March 25-27.

Menzies T.J., Compton P., Mahidadia A., (1992b) *Communicating Research Models of Human Physiology using Qualitative Compartmental Modelling* in **Communicating Scientific and Technical Knowledge**, a AAAI '92 workshop, San Jose, California, July 12-17.

Menzies T., Mahidadia A., Compton P. (1992c) *Causal Explanations as a Tool for Refining Qualitative Models* in the 1992 ECAI workshop: **Improving the Use of Knowledge-Based Systems with Explanation** Vienna August.

Miller P.L. (1986) **Expert Critiquing Systems** Sprinter-Verlag.

Moberg D. (1992) Personal communication.

Neches R. (1991) *Acquisition of Knowledge for Sharing and Reuse* in the **6th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop**, Banff, Canada, October 6-11, pp 18-1 to 18-20.

Patzer K., Sticklen J., Pautisan K. & Robertson G.P. (1992)*Causal Knowledge Sharing in Large Biological Projects: Leveraging the Functional Approach* in **Proceedings of the AAAI '92 Workshop on Communicating Scientific Knowledge**, San Jose, California, July 15.

Pearl J. & Verma T.S. (1991) *A Theory of Inferred Causation* in **Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference** Allan J.A. , Fikes R., Sandewall E. (eds), Morgan Kaufmann (April), CA, pp441-452.

Popper K.R. (1963) **Conjectures and Refutations**, London: Routledge and Kegan Paul.

Preece A.D. & Shinghal R. (1992) *Verifying Knowledge Bases by Anomaly Detection: An Experience Report*, in **Proceedings of ECAI '92**, pp835-839.

Rajogopalan R. (1986) *On Combining Expert System and Qualitative Reasoning Techniques* in **Coupling Symbolic and Numerical Computing in Expert Systems** Kowalik J.S. (ed), Elsevier Science Publishers B.V. (North-Holland).

Rappaport A.T. (1991) *A Theory of Local and Interstitial Knowledge*, in the **6th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop**, Banff, Canada, October 6-11, pp21-1 to 21-18.

Shapiro E.Y (1981) *Inductive Inferenc of Theories from Facts*, **Technical Report 192, Yale University**.

Shaw M.L.G. (1988)*Validation in a Knowledge Acquisition System with Multiple Experts* in **Proceedings of the International Conference on Fifth Generation Computer Systems**, 1988, pp1259-1266

Silverman B.G. (1992)*Survey of Expert Critiquing Systems: Practical and Theoretical Frontiers* **Communications of the ACM**, April , Vol 35, No 4, 106-127.

Simon H.A. (1952) *On the Definition of the Causal Relationship* in **Journal Philosophy**, 49, pp517-528.

Skuce D., Shenkang W. & Beauville Y. (1989) *A Generic Knowledge Acquisition Environment for Conceptual and Ontological Analysis* in the **4th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge Based Systems Workshop**, Banff, Canada, October, pp 31-1 to 31-20

Smythe G.A., Duncam M.W., Bradshaw J.E.,Cai W.Y. & Symons R.G. (1982) *Serotinergic Control of Growth Hormone Secretion: Hypothalmic Dopamine, Norepinephrine and Serotonin Levels and Metabolism in Three Hyposomatrophic Rat Models and in Normal Rats* in **Endocrinology** 110(2), pp376-383.

Smythe G.A. (1987) *Hypothalamic noradrenergic activation of stress induced Adrenocorticotropin (ACTH) release: Effects of acute and chronic dexamethasone pre-treatment in the rat*,in: **Exp. Clin. Endocrinol. (Life Sci. Adv.)**, 6: 141-144.

Smythe G.A. (1989) *Brain-hypothalmus, Pituitary and the Endocrine Pancreas* in Smamols R. (ed) **The Endocrine Pancreas**, Raven Press, New York.

Srinivasan A., Compton P., Malor R., Edwards G. & Lazarus L. (1991), *Knowledge Acquisition in Context for a Complex Domain* in **Proceedings of the Fifth European Knowledge Acquisition Workshop** Weilinga B., Boose J. & Gaines B. (eds), Pergamon.

Steels L. (1990) *Components of Expertise* in **AI Magazine**, Summer, pp29-49.

Struss P. (1992) *Knowledge-based Diagnosis- An Important Challenge and Touchstone for AI*, in **Proceedings of ECAI '92**, pp863-874.

Tu S., Shahar Y., Dawes J., Winkles J., Puetra A., Musen M. (1991) *A Problem-Solving Model of Episodic Skeletal-Plan Refinement* in the **6th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop**, Banff, Canada, October 6-11, 1991, pp 34-1 to 34-20.

APPENDIX: Casual- An Explanation-Orientated Refutation Algorithm of Causal Models Using "Breadcrumbs"

A.1. Implementation Notes

1. To explain all the steady changes, we assume that they are steady and try to explain all the non-steady changes. Internally, this means we copy all the steady nodes to the Done set.
2. Within a logic programming formalism, generating detailed records of failures is intricate. On failure, the program backtracks and unbinds all its knowledge of the failed pathway. Hence, we can add items to the *CantDo* set only in our top-level loop (see *casual1/8*) The knowledge of failed explanations of measured deltas in the recursive calls (i.e. in *casual2/12*) is not accessible.

A.2. Code Listing

```
casual(Effects,Causes,Done,CantDo,Assumes) :-
     extractSteadies(Effects,[],ToDo,[],Stable),
     casual1(ToDo,Stable,[],[],
             Causes,Done,CantDo,Assumes).

extractSteadies([],Changes,Changes,Stable,Stable).
extractSteadies([Node(std)|Effects],
     Changes0,Changes,Stable0,Stable) :- !,
     extractSteadies(Effects,Changes0,Changes,
                     [Node(std)|Stable0],Stable).
extractSteadies([Change|Effects],Changes0,Changes,
          Stable0,Stable) :-
     extractSteadies(Effects,[Change|Changes0],
                     Changes,Stable0,Stable).

casual1([],Done,CantDo,Assume,_,Done,CantDo,Assume).
casual1(ToDo0,Done0,CantDo0,Assume0,
        Causes,Done,CantDo,Assume) :-
     aMember(Node(Change),ToDo0,ToDo1),
     trace(['Analysing ',Node(Change)]),
     (casual2(Node(Change),Node(Change),Done0,
              CantDo0,Assume0,Causes,[Node],
              ToDo1,ToDo,Done1,CantDo1,Assume1)
     ->   trace('Q.E.D.'),
          casual1(ToDo,[Node(Change)|one1],
                  CantDo1,Assume1,
                  Causes,Done,CantDo,Assume)
     |    trace('???'),
          casual1(ToDo1,Done0,
                  [Node(Change) | CantDo0],Assume0,
                  Causes,Done,CantDo,Assume)).

aMember(X,[X|L],L).
aMember(X,[H|Tail],[H|Rest]) :-
aMember(X,Tail,Rest).

% check for breadcrumb that denotes past failure on
% this route
casual2(Node(_),_,_,CantDo0,_,_,Visited,_,_,_,_,_)
 :-
     member(Node(_),CantDo0),!,
     trace(Visited,[Node,' failed above']),
     fail.

% check for breadcrumb that denotes a previous
% explanation of this change
casual2(Node(Change),_,Done,CantDo,Assume,_,Visited,
     ToDo,ToDo,Done,CantDo,Assume) :-
     member(Node(C),Done), !,
     Change = C,
     trace(Visited,[Node(Change),
          ' was explained above']).

% check if we have arrived at a known exogenous
% source
casual2(Node(Change),_,Done,CantDo,Assume,Causes,
        Visited,ToDo,ToDo,Done,CantDo,Assume) :-
     member(Node(C),Causes), !,
     Change = C,
     trace(Visited,[Node(Change),
                    ' is a known cause']).

% check for breadcrumb that denotes a
% previous assumption
casual2(Node(Change),Goal,Done,CantDo,Assume,_,
        Visited,ToDo,ToDo,Done,CantDo,Assume) :-
     member(Node(C),Assume), !,
     (Change = C
     ->   trace(Visited,[Node(Change),
                ' is an existing assumption'])
     |    trace(Visited,[Node(Change),
          ' fails due to assumption clash:',Node(C)]),
          fail).

% if the desired change is one of the currently
% unanalysed measured deltas, recurse.
casual2(Node(Change),Goal,Done0,CantDo0,Assume0,
          Causes,Visited,ToDo0,ToDo,
          [Node(Change)|Done1],CantDo,Assume)   :-
     not(Node(Change) = Goal),
     aMember(Node(C),ToDo0,ToDo1),!,
     C = Change,
     trace(Visited,['Analysing ',Node(Change)]),
     casual2(Node(Change),Node(Change),Done0,
               CantDo0,Assume0,Causes,Visited,
               ToDo1,ToDo,Done1,CantDo,Assume),
     trace(Visited,'Q.E.D.').

% when all else fails, propagate
casual2(Node0(Change0),Goal,Done0,CantDo0,Assume0,
    Causes,Visited,ToDo0,ToDo,Done,CantDo,Assume) :-
     1(Node1,Node0,I),
     noLoop(Node1,Visited),
     applyInfluenceFunction(Change0,I,Change1),
     trace(Visited,['trying ',Node1(Change1)]),
     (Goal = Node0(Change0)
          -> Assume1 = Assume0
          |  Assume1 = [Node0(Change0)|Assume0]),
     casual2(Node1(Change1),Goal,Done0,CantDo0,
             Assume1,Causes,[Node1|Visited],
             ToDo0,ToDo,Done,CantDo,Assume).

noLoop(N,Old) :-
     member(N,Old),!,
     trace(Old,['can''t revisit ',N]),fail.
noLoop(_,_).

% applyInfluenceFunction(Cause,Influence,Effect)
applyInfluenceFunction(up,direct,up).
applyInfluenceFunction(down,direct,down).
applyInfluenceFunction(up,inverse,down).
applyInfluenceFunction(down,inverse,up).
```

Adaptation of a task specific problem solving method to the features of an application*

Klas Orsvärn
Swedish Institute of Computer Science (SICS)
Knowledge Based Systems Laboratory
PO Box 1263
164 28 Kista
Sweden
klasorsv@sics.se

August 17, 1992

Abstract

This paper describes a case-study of how a task specific problem solving method can be adapted to the features of a real application. The starting point is an initial version based on the interpretation model of Systematic Diagnosis by Localization [Breuker et al, 87]. The application is a KBS that guides a service technician in the task of diagnosing a specific measuring instrument. Some features of the application require adaptation of the problem solving method. It does not give enough support for reducing the time required for observations, and it is based on the assumption that parts can be tested independently of other parts. I describe a number of refinements and modifications to the problem solving method that eliminate some of these restrictions.

1 Introduction

In this paper I present an example of how a task specific problem solving method can be refined and adapted to cope with the features of a real application. The goal of the work described here is to provide an empirical basis for:

- developing reusable task specific problem solving methods that give strong support for knowledge acquisition by being based on strong assumptions about the application.

*The research reported here was carried out in the course of the KADS-II project. This project is partially funded by the Esprit Programme of the Commission of the European Communities as project number 5248. The partners in this project are Cap Gemini Innovation (F), Cap Programator (S), Netherlands Energy Research Foundation ECN (NL), ERITEL SA (ESP), IBM France (F), Lloyd's Register (UK), Swedish Institute of Computer Science (S), Siemens AG (D), Touche Ross MC (UK), University of Amsterdam (NL) and Free University of Brussels (B). This paper reflects the opinions of the author and not necessarily those of the consortium.

- investigating principles for refining, combining, and adapting such methods to the features of a real application.

Instead of developing a new application for this purpose, this example is the result of reverse engineering from an existing KBS that guides a service technician in the task of diagnosing a specific device, which is a measuring instrument for testing telecommunications equipment. The application is henceforth called HP. It is a consultation system that makes a sequence of requests for tests to be performed by the user, who inputs the results of each test to the system. The system finally reports which part of the device is faulty.

The KBS was developed in 1989 by Infologics and Swedish Telecom. No task specific problem solving methods were used in developing the KBS. It is specified with a decision tree (with some extensions) and implemented as a rule-based system.

The goal of the reverse engineering was to specify the system as an application of task specific problem solving methods. The interpretation model of Systematic Diagnosis by Localization [Breuker et al, 1987], [Wielinga et al. 1992] was used as a starting point, since it captures a central strategy used in HP, which is to narrow down the location of the fault iteratively by dividing the device hierarchically into parts which are tested separately.

Not all aspects of the behaviour of HP could be specified in terms of Systematic Diagnosis by Localization. Therefore I needed to adapt and refine the problem solving method. I did that by trying to provide sufficiently general explanations for the deviations in behaviour. The explanations were created by studying technical documentation of the HP device, and by interviewing the knowledge engineer and domain expert who developed the system. In this process I found a number of weaknesses in the system, that essentially correspond to false assumptions. The reason why they have not been discovered before is partly because they are left tacit in the original specification and implementation, and partly because the system has not been extensively used. These discoveries can be used to demonstrate advantages of using explicit task specific problem solving methods for purpose of validation, although that is beyond the scope of this paper, but they made it more difficult to derive a task specific problem solving method. Nevertheless, I have found a number of useful strategies. Since the strategies were only tacitly applied, they were also applied inconsistently.

The adaptations to the problem solving method that are described here are based on my conclusions about what the KBS could have looked like if it had been based on consistent application of task specific problem solving methods. I have tried to make it as faithful as possible to the existing KBS. Work is in progress to re-specify the KBS as an application of the set of methods proposed here. The result will then be evaluated by the domain expert who participated in development of the original KBS.

After a brief description of the modelling language used, I will describe the initial version of the problem solving method, which is my interpretation of Systematic Diagnosis by Localization. To avoid confusion with the original description of Systematic Diagnosis, I will use the name Top-Down Fault-Localization (TDFL) for the set of task specific problem solving methods presented here.

I will then discuss the assumptions on which the initial TDFL is based, and argue that the method can be rationally justified. After that I will describe the features of the HP application. We will see that those features are not consistent with the assumptions of the initial TDFL. My guess is that this is the case in most application of technical diagnosis. So the initial TDFL cannot be justified as a solution to the HP task. I then describe how the features are handled in HP, and propose a number of revisions that finally result in a version of TDFL that captures most of the behaviour of the HP application. Finally I make some conclusions.

2 The modelling language

The description of TDFL presented here is just an informal specification. It is intended to be interpreted as a knowledge level model of a the KBS [Newell, 82], in that it specifies the knowledge that is used to select external actions in a situation on the basis of the particular goal of troubleshooting. I do not use any particular conceptual modelling language. The models are only based on the simple idea of defining a method to accomplish a task as a decomposition of that task into subtasks. which I see as the unifying idea of interpretation models [Breuker et al, 87], role-limiting methods [McDermott, 88], and method components of expertise [Steels, 90]. This is very similar to the problem solving interpretation of Horn clauses [Kowalski, 79]. I therefore use the Horn clause subset of first order predicate logic to represent this task decomposition. A task can be described as a predicate. The input and output of the task are represented as arguments of the predicate. Usually the input is the first argument and the output is the last. The set of subtasks are represented as the body (the antecedent) of a Horn clause where the head (the consequent) represents the task. If there are alternative decompositions, i.e. alternative methods, they are represented as different clauses. Actions that are taken in the external world, what is called transfer tasks in [Wielinga et al, 92], are also represented as subtasks. However. instead of providing a method for solving that task, it is solved by taking an action in the external world. e.g. by querying the user.

To specify the problem solving behaviour. i.e. the sequence of actions in the external world, we must add a procedural control on the task decomposition. I use a simple way that often works, which is to use the procedural interpretation of Prolog (e.g. [Kowalski, 79]). It simply means that you solve the subtasks one by one in the sequence they are written in the decomposition. It may be the case that a method does not provide an output solution, in which case the subtask fails. A task may have more than one solution, i.e. more than one possible output value. e.g. if there is more than one method. If there is more than one method. they are performed one at a time in the order in the which they are written, until the task is solved. If a subtask fails, you go back to the previous subtask that produced the input to the failed subtask and try to produce a new input to retry the failed subtask, etc. In order to specify iteration, we must then use recursive decompositions. I have found it convenient to extend the Horn clause language with an if-then-else construct. If the precondition subtask succeeds, you continue with the postcondition subtask. otherwise you continue with the subtask in the else-part. If the postcondition fails. you may go back and try more solutions of the precondition.

In the models, I only break down tasks into subtasks until I get to a level where the task can be viewed as (case independent) domain knowledge, and there is no longer any need for procedural control. At the lowest level, I define the tasks verbally. My choice of where to stop decomposing is rather arbitrary. Some of the tasks could easily be further decomposed. I actually decompose the tasks more than what is necessary for controlling the sequence of external actions, so there is more control in the models than what is desirable in a knowledge level model.

3 Initial version of TDFL

The initial version of TDFL is based on my interpretation of Systematic Diagnosis by Localization [Breuker et al. 87], [Wielinga et al. 92]. I have made some immediate modifications to reduce the differences to the revised versions that I will present subsequently. What is said about this initial version is therefore not necessarily true for the original Systematic Diagnosis by Localization. If anything, my initial version is more restricted than then original version.

The top level task is called *troubleshooting*. The method is defined in figure 1, which also contains the methods of the subtasks. The subtasks which are not further decomposed are defined verbally below. The input to the task is a reference to a device. There is no information about what is wrong with it. The device has a number of distinct functions that can be used independently. The HP device can for example be used to control the amplitude and frequency of an output signal. It can also be used to measure amplitude and frequency of an input signal. The first thing to do is to find out which function is faulty. Each function is tested until one is found to be faulty, which is when the refute subtask succeeds. We have then detected a specific fault.

Diagnosis is the task of localizing the fault to the smallest possible level. We do that iteratively to a lower and lower level of detail. I distinguish between topDownDiagnose, which is the task to find the most detailed description of the fault, and diagnose, which is the task to find a more detailed description of the fault. The refute task takes a specified norm as input, which plays the role of hypothesis. We then need to know what observations must be done to determine if the norm is satisfied. This is basically an instruction to the user. The next subtask is a transfer task. The user makes the required observations, which is a description of the actual behaviour of the part. If the actual behaviour is inconsistent with the specified norm, the part is faulty. The topDownDiagnose task is simply the transitive closure of the diagnose task. The first step in the diagnose task is to decompose the part corresponding to the fault into a hypothesis part and the remaining parts which are called the rest. The hypothesis part is tested with the refute subtask. If the test fails, we make the conclusion by elimination that the fault is in the rest.

deviceFunction(*Device*, *Function*): A device often consists of several distinct functions that are used individually. Each of those device functions is a a solution to this subtask. This subtask is only applied to the top level device. The device functions may be overlapping, in the sense that there may be a lower level part that is used to accomplish two

```
troubleshooting(Device, LowestLevelFault) —
    deviceFunction(Device, Function)∧
    specify(Function, Norm)∧
    refute(Norm)∧
    topDownDiagnose(Function, LowestLevelFault)

refute(Hypothesis) —
    observables(Hypothesis, Observables)∧
    observe(Observables, Observation)∧
    notconsistent(Observation, Hypothesis)∧

topDownDiagnose(Fault, LowestLevelFault) —
    diagnose(Fault, RefinedFault)∧
    if topDownDiagnose(RefinedFault, RefinedFault2)
        then LowestLevelFault = RefinedFault2
        else LowestLevelFault = RefinedFault

diagnose(FaultyPart, RefinedFault) ←
    decompose(FaultyPart, HypothesisPart, Rest)∧
    specify(HypothesisPart, Norm)∧
    if refute(Norm)
        then RefinedFault = HypothesisPart
        else RefinedFault = Rest
```

Figure 1: The initial version of methods used in TDFL.

device functions.

decompose(*SuperPart*, *SubPart*, *Rest*): SuperPart is a part that consists of the two parts SubPart and Rest. There may be many ways of decomposing a part into two subparts, i.e. many solutions to this subtask. The two output parts may in turn be decomposed into parts in the same way. The domain model of the device does not have to be a binary decomposition. This task can be defined in terms of any kind of partOf-model. Rest may then be a reference to a set of parts.

specify(*Part*, *Norm*): Norm is a set of necessary and sufficient conditions for Part to be functioning correctly

observables(*Norm*, *Observables*): We have to look at the norm and see what observations need to be done in order to test whether the norm is satisfied. The Observables are the attributes which are referred to in the Norm.

observe(*Observables*, *Observation*): This is a transfer task, i.e. it is solved by an external action. The output observation is the actual observation made of the observables by the user.

consistent(A, B): A and B are expressions. In TDFL, A will be a norm and B will be an observation. The subtask succeeds iff the norm is consistent with the observation.

4 Rational Justification

Although the above version of TDFL is extremely simple, it captures an important method used by service technicians performing the kind of task that the HP application is to support. They use other methods as well, but this one requires less experience of the particular device. In applications like HP, a service technician may never have seen the device before, but uses technical documentation to systematically decompose the device into parts and specify norms that can be tested. The term Systematic Diagnosis is in that sense quite suitable.

This version of TDFL has the attractive feature that we can rationally justify that it works.[1] Given a definition of the goal and a set of reliable assumptions[2], it is possible to prove that the method works. One might argue that this should be required of a knowledge level model. The principle of rationality says that an agent takes an action if it knows that the action will lead to the goal [Newell, 82]. So this knowledge must be part of a knowledge level model.

Task definition The input is a device. The output is a faulty part of the device. The goal is to find a faulty part of the device. It is desirable to locate a part at a fine level of detail.

Assumptions Domain knowledge assumptions: the agent knows every part of the device, and the parts of every part, and the agent knows the correct behaviour of each part. Transfer task assumption: if requested, the user will tell the agent the actual behaviour of any part. Case assumption: there is a fault in a part of the device. Method assumptions: a part behaves incorrectly if and only if there is a fault in that part, and faults are persistent, i.e. not intermittent.

Justification The agent will find a faulty part of the device, because one of the parts is faulty and the agent will find out which one by noticing the difference between actual behaviour and correct behaviour. By diagnosing the faulty part in turn, the agent will to the best of its ability locate the fault to a finer level of detail.

[1] The term rational justification was used in the context of validation by [Steels, 91].

[2] I use the term reliable assumption here in in the sense that we can expect a domain expert to know whether the assumption holds or not.

Note that we don't need to make a single-fault assumption. This is partly because the goal is only to find one fault. If there are more faults, you have to diagnose again. If we would have used a fault-model. the single fault assumption would have been necessary, because multiple faults may change the effects of each other. It is then the domain knowledge that is based on the single fault assumption. specifically that the agent knows the effect of a fault.

Other assumptions of Systematic Diagnosis have been mentioned in [Duursma, 1992]. For example that the cost of observing the behaviour of a part is equal for all parts. This assumption is necessary if a constraint is to keep down the cost of observations. You might argue that it is a tacit assumption of Systematic Diagnosis and the initial version of TDFL that the cost of observations should be kept down, because otherwise there would be no reason to decompose the device iteratively. If so. the task definition should contain this constraint. But if we do want to keep down the cost of observations, the agent must also have the ability to keep down the number of observations. I will get back to this after discussing the features of the HP task.

5 Features of the HP application

The following is a description of the features of the task and domain in the HP application. We will see that the initial TDFL is too simple to handle them. Observations must be made by the user, and they take time. One constraint is therefore to keep down the number of observations that need to be made in order to diagnose. More precisely, it is the total time required for observations that we want to keep down. Some observations take more time than others. A sometimes conflicting constraint is to choose a sequence of actions that the user will be able to understand. Another requirement is that the fault should be located to an appropriate level of detail. It should not be too low, because it requires more work from the user who makes observations. and it may be difficult to repair at a too detailed level.

We can distinguish some significant characteristics of the particular HP device, although they apply to most electronic devices. Most parts *cannot* be tested completely independently of the rest of the device. In particular you cannot control the inputs of a part with anything else than the external controls of the device. Although you can in principle force a signal directly on an internal conductor. you do not want to, because you may destroy something in the process. If you want to control an input of a component which is internal to the device, you have to input an external signal to the whole device and let it go through other components until it reaches the one you want. You have to make sure that the signal has not been corrupted on the way. Some parts are mounted so that they can be easily removed and tested independently, but that may also cause damage, and it is more the exception than the rule. Due to feedback loops, some inputs of some parts cannot be controlled at all. and hence those parts cannot be directly tested.

Recalling the assumptions of the initial TDFL. we must also make clear that the behaviour of an electronic device and its parts is too complex to be observed in its whole. It is however possible to specify norms that can be tested and are sufficiently strong to detect

most faults.

In the rest of the paper, I will describe how HP handles these features, and what adaptations I have made in TDFL in order to accomplish this behaviour. I will divide the description in two parts. The first part deals with the constraint to keep down the time required for observations. The second part deals with the limited testability.

6 Reducing the time required for observations

The sequence of tests in HP reflects a strategy to keep down the total time required for observations in order to reach a final diagnosis. This constraint is not well defined. A judgement has to be made in each application about how much needs to be done.

To a great extent, this behaviour of HP can be described in terms of TDFL as a decomposition model tailored to diagnosis. In the next section, I will describe some features of the resulting decomposition in HP. Those features do not require any modifications to TDFL, but correspond to necessary assumptions about the domain knowledge.

The sequence of tests can however not be described only in terms of a static decomposition model in TDFL. In some cases the choice of which observation to make next is dependent on the actual faulty observation, rather than just its inconsistency with the norm. The explanation is that a particular part is more likely than the others to cause this particular fault. In order to describe this behaviour, the actual faulty behaviour must be input to the diagnose task, rather than just a reference to the faulty device function. In addition, the method of the decompose task must make use of causal associations and select hypothesis on the basis of the actual input fault. These modifications to TDFL will be described in subsequent sections.

HP never requests the user to observe the whole specified function of a the device or a part. Most functions are tested with only one input value. This corresponds in TDFL to specifying a very limited norm. In order to justify this, we can make the assumption that a fault will currupt all outputs of the function. Actually, the real reason is that HP is not required to detect every fault.

There are cases in which several input values are used in HP to test a function. The explanation is that different parts of the device are used to produce the behaviour depending on what the input value is, i.e. there is some kind of switch. This means that if only one of the input values generates a faulty output, the fault must be located in the part that is used for only this value. If all values generate faulty behaviour, a single-fault assumption leads to the conclusion that the fault must be in parts that are used for all input values.

In one case, the different input values are used when a device function is first tested in order to detect a fault. In other cases, different input values are used long after detecting the fault, in order to locate the fault within a part that has been found to be faulty.

This technique is however not used consistently in HP. It is only used in a few cases, and there are many more cases in which it would be just as reasonable to use it. It seems that

a method for handling this in all desirable ways would be quite complicated, and I have therefore postponed it. The easiest solution would probably be to employ a causal fault model for this, as for example in Cover-and-Differentiate [Eshelman, 88].

6.1 Decomposition model tailored to diagnosis

The only support that the initial version of TDFL provides for reducing the time required for observations is that the device is decomposed iteratively. This means that we never have to observe all parts at the lowest level of decomposition. By observing a composite part, we can exclude all the primitive parts within it. If we assume that it is cheaper to test a composite part than to test its parts, this will be a way to keep down the total time required for observations. However, iterative decomposition is in itself a very weak way – so weak that it will never be enough. In practice we make tacit assumptions about the decomposition model of the device, e.g. that a top level function is not decomposed directly into primitive parts. If TDFL is to satisfy the needs to reduce the time required for observations, we must either tailor make the decomposition model to diagnosis, or provide methods for doing this. By decomposition model I mean the domain knowledge that is used to realize the tasks deviceFunction and decompose that have been used in the initial version of TDFL.

If we assume that the level of primitive components is given, the decomposition of a device into primitive components is well defined. However, in iterative decomposition we want to introduce intermediate levels of decomposition – composite parts. The choice of what to regard as a composite part is arbitrary, unless we provide a more constrained definition. The concept of composite part in itself does not even imply a hierarchical decomposition, although there are reasons to prefer such a decomposition.

Although the choice of decomposition model is not well defined, there are some natural foundations. One of them is the decomposition made in the design of the device. Another is the physical configuration. A composite physical part must be such that it can be physically removed as a whole from the other parts. Another kind of constraint on a composite part may be that the subparts should be functionally part of the composite part. In diagnosis we want a functional decomposition, since it is the functions we are testing, and we want primitive parts that are physical parts, so that they can be replaced or repaired. However, those constraints are not enough. If it is cheaper to test a composite part than it is to test its subparts, then there is reason to include it in the decomposition. The cost of locating the fault inside a composite part, if the hypothesis is confirmed, should be equal to the cost of locating the fault in the remaining set of suspect parts, if the hypothesis is refuted.

Another important tacit assumption about the decomposition model is that the parts have inputs that can be controlled and outputs that can be observed during diagnosis. Altough the decomposition should be functional, the observations are made of the physical device. The sequence of observations, which is controlled by the decomposition model in TDFL, should therefore be such that consecutive observations are physically close, e.g. on the same circuit board. This means that the cost of testing a part is context dependent.

Ideally we should provide a method that creates a good decomposition for diagnosis on the basis of less task dependent device models. However, I think it is difficult, since there are so many factors to take into account. In the absense of such a method, we have to assume that a service technician can develop a decomposition that is well enough tailored to diagnosis.

7 Detecting characteristics of the fault

The test subtask in the initial version of TDFL only results in success or failure. In the test method, we actually find out more. We find out the actual behaviour of the faulty part. This knowledge can be used to reduce the time required for observations in the subsequent process of finding the lowest level fault.

One way is to recognize the effects of some specific faults, and use causal association to select more likely hypotheses. We can incorporate this change by first modifying the methods of troubleshooting and diagnose. so that the input to diagnose is not just a reference to a faulty part, but a reference to the actual faulty behaviour of that part. Since the output of diagnose becomes the input to a subsequent invocation of diagnose, the output must also be a reference to the actual faulty behaviour of the lower level part.

7.1 Fault as the actual faulty behaviour

We must first make the notion of behaviour a little bit more precise by elaborating the verbal definition of the subtask specify.

specify(*Part*, *Norm*): As in the initial version. Norm is a set of necessary and sufficient conditions for Part to be functioning correctly. More precisely, it is a description of the correct behaviour of a part of a device. The behaviour can be described as a mathematical function, which maps a set of input values to an output value.

The revised version of the method of troubleshooting is described in figure 3. The only difference is that the refute task outputs the actual faulty behaviour, which is then input to topDownDiagnose. The new refute method is described in figure 2.

refute(*Hypothesis*.*Observation*) ←
    observables(*Hypothesis*.*Observables*)∧
    observe(*Observables*.*Observation*)∧
    notconsistent(*Observation*.*Hypothesis*)∧

Figure 2: The refute method with the actual observed faulty behaviour as output.

The subtask decompose in the method of diagnose should now take the faulty behaviour of

$troubleshooting(Device, LowestLevelFault) \leftarrow$
$deviceFunction(Device, Function) \wedge$
$specify(Function, Norm) \wedge$
$refute(Norm, ActualBehaviour) \wedge$
$topDownDiagnose(ActualBehaviour, LowestLevelFault)$

Figure 3: The method of troubleshooting adapted to hypothesis as fault.

a part as input. The output should be hypotheses about the behaviour of lower level parts. Since it is a slightly different task defined in terms of the previous version of decompose, I will give it a new name, decomposeB. As we will see later, it would become awkward to assign meaningful names to refined versions of the task.

decomposeB($Fault, Hypothesis, Rest$): Fault is the behaviour of a part. The part corresponding to Fault is decomposed using the previous decompose subtask into the parts corresponding to Hypothesis and Rest. Both Hypothesis and Rest are statements about behaviours. One of the two parts shares output with the part corresponding to Fault. If that part is faulty, we know that it must produce the same faulty output produced by the part corresponding to Fault. The hypothesis for that part is then that it has precisely this behaviour. For the other part, the hypothesis is the negation of the statement that it has the specific behaviour which is its norm. Typically, the decomposition will be such that the selected hypothesis is a part that shares input with the decomposed part, and the rest is a single part that shares output with the decomposed part, but that is influenced by testability, which is discussed in a later refinement.

In contrast with the initial version of diagnose, the hypothesis now refers to fault rather than correctness. We therefore want to confirm rather than refute it. The confirm method is identical to the refute method, except for the last condition that it succeeds when the hypothesis is consistent with the observation, and I will therefore not write it down here explicitly.

In the version of the diagnose method in figure 4, the fault is the actual behaviour of the faulty part. Since the fault is a behaviour rather than a part, it does not make sense to decompose it into parts. Instead the fault is decomposed into hypotheses about faults of parts. The hypothesis is then a statement that can be verified. If the hypothesis is not confirmed, we make the conclusion by elimination that the part corresponding to Rest is faulty.

8 Testing more likely hypotheses first

Now that the actual faulty behaviour is input to diagnose, more likely hypotheses can be detected by making causal associations, and thus generated before less likely hypotheses.

$diagnose(Fault, RefinedFault) \leftarrow$
$decomposeB(Fault, Hypothesis, RestHypothesis) \wedge$
$if\ confirm(Hypothesis, ActualBehavior)$
$then\ RefinedFault = ActualBehaviour$
$else\ RefinedFault = RestHypothesis$

Figure 4: The method of diagnose with the actual behaviour as fault

The new method of decompose is defined in terms of the previous version, with the additional constraint to use the task *preferInSituation* to first select a likely hypothesis. The methods of decomposeBP and preferInSituation are described in figure 5. Note that the causal association is only used to select hypothesis. The subsequent diagnosis is still based on the decomposition model.

$decomposeBP(Fault, Hypothesis, Rest) \leftarrow$
$preferInSituation(Fault, Hypothesis) \wedge$
$decomposeB(Fault, Hypothesis, Rest)$

$preferInSituation(Fault, Hypothesis) \leftarrow$
$partOf(Fault, Hypothesis) \wedge$
$causes(Hypothesis, Effect) \wedge$
$consistent(Effect, Fault)$
$\vee$
$directPartOf(Fault, Hypothesis)$

Figure 5: The method of decomposeBP based on selecting the most likely hypothesis, and the methods of its subtask preferInSituation

We have to distinguish between parts in general and the direct part in a tree-like decomposition. This is a distinction which was previously hidden in the verbal description of the first version of decompose.

$directPartOf(SuperPart, SubPart)$: This relation means that SubPart is a direct part of SuperPart in the decomposition model of the device. For simplicity, SuperPart and SubPart may also refer to a behaviour corresponding to that part.

$partOf(SuperPart, SubPart)$: This is the transitive closure to the relation directPartOf. In other words, if X is a direct part of Y, then X is also a part of Y, and if X is a part of Y, and Y is a direct part of Z, then X is also a part of Z.

$causes(Fault, Effect)$: The relation means that the fault always results in the effect, if there are no other faults that influence the effect. This causal relationship is limited to

those causal relationships that a service technician would use to override the more systematic diagnosis by decompose. It is limited to a few well known causes with very specific effects. For example, in the HP application one specific component has the property that if it breaks, the amplitude of the output signal of the device will be reduced by half. Fault is the behaviour of a part. It is typically negatively defined, i.e. the statement that the norm of the part is not satisfied. The part corresponding to Fault must be a part of the part corresponding to Effect.

9 Hypotheses that are not directly testable

The next application feature to deal with is that a part cannot be tested unless the preceding parts in the signal flow are correct. This is because signals can only be input at the external inputs of the device. In other words, the assumption that the user can observe the behaviour of any part is violated. There are two quite different examples of this. The worst example is a feedback loop, i.e. an input to a part is connected directly or indirectly to an output of the same part. The other example is when the input is on an acyclic path connected to an input of the device. Feedback loops are not handled well in HP. I will therefore not present a method for handling those, but I will briefly describe a possible limited solution inspired by HP. First I will describe the acyclic example in more detail, and provide a method that handles it.

9.1 Preceding parts have not been tested

Imagine that we have located the fault to a block of components that are connected together in a single signal chain, as in figure 6. The only component that we can test directly is the first component in the chain, the one that has the same input as the chain. For any other component, the input will be unpredictable if the fault is earlier in the chain. Another way to put it is that we test a component by measuring its outputs. We must already know the input to a component when we test it.

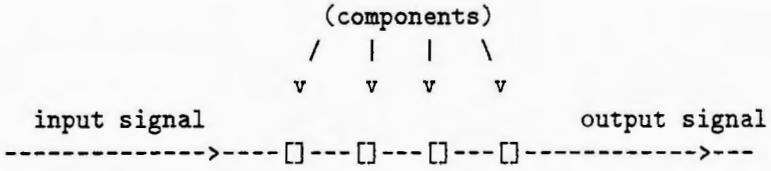


Figure 6: Four components in a signal chain

This means that we cannot generate hypotheses in the same random way as before. It would be relatively easy to modify the decompose step so that hypotheses are generated in the order dictated by the signal flow. However, this does not work if we want to give

priority to more likely hypotheses, e.g. those causally associated with the actual fault. Unless the hypothesized part is first in the signal flow, the hypothesis will not be directly testable. In HP, this problem is actually not solved. The causal association leads directly to a conclusion, although it is not necessarily correct. The user then has to continue diagnosis without the help of the KBS. The following describes the strategy that would have been needed in order to support the test of this kind of conclusion.

Let us assume again the example of a fault located to a chain of four components. The hypothesis is that the third component in the signal flow is faulty. What we do then is measure the output of the third component, knowing the input to the chain. If that observation is ok, we have refuted the hypothesis. If it is not ok, we have still not confirmed the hypothesis. We must also measure the input to the third component. If that is ok, we have confirmed the hypothesis, and we may try to refine it. However, if the observation is not ok, we have confirmed another hypothesis, namely that the fault is in the 1st or 2nd component.

A problem with describing a method for testing a hypothesis that is not directly testable is that the process of testing one hypothesis may result in other hypotheses being confirmed or rejected. A natural way to deal with this is to describe it as a method of test by elimination. We have to identify other hypotheses that can be tested and that can be used to make a conclusion by elimination about the desired hypothesis. You might say that we have to make a plan. In this application it is enough to identify two auxiliary hypotheses. The first one is the subsuming hypothesis. It is the smallest part that both contains the desired part and is testable. This will be a part that shares the outputs with the hypothesis part and which inputs are also inputs of the faulty part, and therefore have been observed. The next auxiliary hypothesis is the rest of the subsuming part minus the desired hypothesis part. It will be testable since its inputs are also inputs of the faulty part.

9.2 Generating auxiliary hypotheses for elimination

Then we arrive at the final formulation of diagnose in figure 9. It introduces three subtask. The subtask called *testable* is verbally defined below. The method of *generateElimination*, described in figure 7, takes the actual fault and the desired hypothesis as input, and generates intermediate hypotheses that are testable and can be used to establish the desired hypothesis by elimination. The faulty part is decomposed into a subsuming hypothesis and the rest, called LargerHyp and RestHyp1. The subsuming hypothesis is decomposed into the desired hypothesis and RestHyp2. A rest may always refer to a set of parts. In previous versions of diagnose, it has never been necessary to observe the behaviour corresponding to a rest-hypothesis. Now that we need to do that, we do not want to use the confirm method. The reason is that we only need to confirm one of the hypotheses in the set. The confirmSet method in figure 8 takes care of that. We should also modify the method of preferInSituation, to prefer testable hypotheses if no other preferences are applicable, but I leave the description of that modification out of this paper.

testable(*Hypothesis*, *Fault*): This test succeeds iff the input attributes of Hypothesis are

$generateElimination(Fault, Hypothesis, LargerHypothesis, Rest1, Rest2) \leftarrow$
$decomposeBP(Fault, LargerHypothesis, Rest1) \wedge$
$testable(LargerHypothesis, Fault) \wedge$
$decomposeBP(LargerHypothesis, Rest2, Hypothesis) \wedge$
$testable(Rest2, LargerHypothesis)$

Figure 7: The method of the task generateElimination

$confirmSet(SetOfHypotheses, Observation) \leftarrow$
$member(Hypothesis, SetOfHypotheses) \wedge$
$observables(Hypothesis, Observables) \wedge$
$observe(Observables, Observation) \wedge$
$consistent(Observation, Hypothesis) \wedge$

Figure 8: The confirmSet method that outputs the observation that confirms one of the hypotheses is the input set.

also input attributes of Fault. The meaning of testable with a negatively defined fault is that the hypothesis would be testable if the fault had been observed.

The neat distinction between generate and test is disturbed in this version of the diagnose method. It may be possible to remedy this by replacing decompose with a task that generates a test plan. In this version, you might say that the method contains two predefined plans. An alternative solution would be to only generate testable hypotheses. To get the desired behaviour with non-testable hypotheses, the decompose method would effectively in each invocation have to recognize that it should generate the next hypothesis in the plan. That would disturb the declarative character of the decompose task.

9.2.1 Feedback loops

In a feedback loop, an input to a part is connected directly or indirectly to an output of the same part. If the feedback input value is faulty, the behaviour corresponding to the specified norm cannot be observed. The only way to solve this in general is to break the loop, e.g. by physically disconnecting it, or using a specific control signal. This possibility is not used in HP. There are a number of methods that would sometimes work when the loop cannot be broken. However, what HP does is simply assume that the feedback input is correct. The only fault that can be detected then is when the fault is located somewhere along the feedback path, before its end. You might wonder how this assumption could possibly hold. The explanation is that feedback loops usually involve components with state. The feedback input might be correct if the component is in the right state. This usually means that the fault in the loop is only a minor deviation from the norm.

```
diagnose(Fault, RefinedFault) ←
   decompose3(Fault, Hypothesis, RestHypothesis)∧
   if testable(Hypothesis, Fault)
      then
         if confirm(Hypothesis, ActualBehavior)
            then RefinedFault = ActualBehaviour
            else RefinedFault = RestHypothesis
      else
         generateElimination(Fault, Hypothesis, LargerHyp, RestHyp1, RestHyp2)∧
         if confirm(LargerHypothesis, LargerBehaviour)
            then
               if confirmSet(Rest2Hyp, Rest2Behaviour)
               then RefinedFault = Rest2Behaviour
               else RefinedFault = Hypothesis
            else
               RefinedFault = RestHyp1
```

Figure 9: The diagnose method adapted to handle hypotheses that are not directly testable.

In order to describe this behaviour in terms of TDFL, we need a way to detect feedback loops, and we need to test the loop by observing if the feedback input is correct. If so, the faulty part of the loop can be diagnosed in the usual way. Otherwise, the method simply fails to locate the fault in the composite part containing the parts in the loop.

10 Conclusions

As I stated in the introduction, the primary goal of the work presented here is to provide an empirical basis for developing reusable task specific problem solving methods and principles for refining, combining, and adapting such methods to the features of a real application. Although I have not yet created a validated specification of the whole KBS, the results so far can still be used to draw some conclusions.

The final version of TDFL presented here is too limited to provide adequate support for knowledge acquisition. HP is a very simple application,which is the reason why we chose it as an example. However, it does contain some examples of more advanced behaviour, which would not be supported by TDFL, and which would be much more common in more advanced applications. The most important limitation, as far as I can see, is the inability to select actions intelligently on the basis of knowledge that different parts process different input values, and knowledge that the same part is used in several functions, which I discussed in section 6.

The other purpose of this example is to investigate principles for refining, combining, and adapting generic methods to the features of a real application. I have not reached any

final conclusions about this, but the following are some suggestions in that direction.

I have presented different versions of methods to reach a very similar goal. From a methodological point of view, it is desirable to have a way to describe what is in common between the capabilities of different methods, and which capability is specific to one method. The approach of [Steels, 90] is to distinguish between a conceptual description of a task, the nature of the input output relation, and the pragmatic constraints of a task. This seems to be a useful distinction in my example. In that sense, what is in common between the different methods is that they accomplish the same task from a conceptual viewpoint. What differs between them is that they satisfy different pragmatic constraints. In the Horn clause representations that I have made, these task feature distinctions have not been made explicit. In the case of the task decompose, I needed to represent the difference somehow, because one method was defined in terms of the another method for the same task. I did that by giving the task a different name, by adding a suffix to the name. In general that leads to redundancy, with different feature combinations. The modelling language should provide a way to make task features explicit. A method that handles a specific feature, should be defined in terms of the task regardless of features. There is clearly a similarity to how specializations are made in object oriented programming.

Unfortunately, the distinction between conceptual and pragmatic features of a task does not seem to be well defined. As a way to capture commonalities and differences between methods, the distinction is relative to a set of methods.

If we want to provide guidelines for this kind of development process, it seems necessary to describe these kinds of refinements and modifications as applications of more general problem solving methods. The whole method of TDFL appears to be based on the same establish-refine regime as Hierarchical classification [Chandrasekaran, 86] and top-down refinement [Steels, 90]. I think they can all be described as applications of binary search, in a general sense, which can be used when it is possible to determine with certainty in which part of the search space the solution is to be found. We must have a way to generate hypotheses that suits the estalish-refine regime. One way is to use a hierarchical decomposition model of the device. Another way is to use a method to create such a decomposition on the basis of other information of the device. The decompose task is also an application of abduction, based on the assumption $partOf(A,B) \wedge not(functioning(B)) \; -\!\!\!\!\!\!\rightarrow \; not(functioning(A))$. Hypotheses are tested by empirical observation. If a hypothesis is negative. the observation tries to refute the negated statement. The observation directly confirms or refutes the hypothesis. Although the observation in practice involves manipulation of external device controls, we can abstract away from that. Some hypotheses cannot be directly tested, in which case they are tested with a plan for test by elimination which is based on a sequence of the more straightforward empirical test.

11 Acknowledgements

I wish to thank Helena Tallqvist of Infologics and Anders Hemström of Swedish Telecom for providing information about the HP application. Cuno Duursma of the Free University

of Brussels for his comments on previous draft papers about this work, and Göran Falkman and Jonas Varnby for pointing out errors in the draft version of this paper.

12 References

Breuker, J. (Ed.) (1987). Model-Driven Knowledge Acquisition: Interpretation Models. Deliverable task A1, Esprit Project 1098.

Chandrasekaran, B. (1986). Generic Tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert.* Vol. 1, No. 3, 23-30.

Eshelman, L. (1988). MOLE: A Knowledge-Acquisition Tool for Cover-and-Differentiate Systems. In S. Marcus (Ed.), *Automating Knowledge Acquisition for Expert Systems.* Kluwer Academic Publishers.

Klinker, G., Bhola, C., Dallemange, G., Marques, D. and McDermott, J. (1991). Usable and reusable programming constructs. *Knowledge Acquisition*, 3(2), June 1991, 117-137.

Kowalski, R. (1979). Logic for Problem Solving. North-Holland.

McDermott, J. (1988). Preliminary Steps Towards a Taxonomy of Problem-Solving Methods. In S. Marcus (Ed.). *Automating Knowledge Acquisition for Expert Systems.* Kluwer Academic Publishers.

Newell, A (1982). The Knowledge Level. *Artificial Intelligence* 18:81- 132.

Steels, L. (1990). Components of expertise. *AI Magazine*, Summer 90.

Steels, L. (1991). Keynote Speech. Eurovav91, European Workshop on the Verification and Validation of Knowledge Based Systems, July 1991, Cambridge, England.

Wielinga, B., Schreiber. A., & Breuker, J. (1992). KADS: A modelling approach to knowledge engineering. *Knowledge Acquisition* 4(1).

Further Research in Task-Specific Integration Architectures for Heterogeneous Problem-Solvers

Dr. W.F. Punch
Knowledge Based Systems Lab, Computer Science Dept.
A714 Wells Hall, Michigan State University
E. Lansing MI 48824
517-353-3541, punch@pleiades.cps.msu.edu

We have previously reported on the TIPS approach to integrated reasoning based on a task-specific view of problem-solving. While TIPS has provided a basis for research on task-specific integration, it lacked a number of features important for a more robust approach to integrated problem-solving. This paper reports on an improved approach to task-specific integration, based both on the original TIPS architecture, new features important for integration and a more formal description of the integration problem itself.

Keywords: Integration, Heterogeneous Agents, Task-specific architectures.

1 Introduction

We have previously described the TIPS (Task Integrated Problem-Solver) architecture (Punch, 1991; Punch and Chandrasekaran, 1990), an embodiment of a task-specific approach to integrated problem-solving. The term task-specific problem-solving has been loosely used for some time, but is generally associated with the work of Chandrasekaran (Chandrasekaran, 1986; Chandrasekaran, 1987). The concept is simple, one expects that the representation of knowledge in a diagnosis system be significantly different from that of a design system. Each such system has special strategies tuned to performing that particular kind of reasoning (diagnosis, design), each has particular representations of domain knowledge to be used by the associated strategies. Each of these *tasks*, associations of knowledge to be used for certain kinds of reasoning, can be realized as a *task-specific architecture*. That is, each task is actually used by developing languages and knowledge representations that encode both the problem-solving strategy and knowledge that is appropriate for solving problems of that

type. These TSA's can form as a building blocks for creating more complicated reasoning systems by appropriate *integration* into the larger context.

As previously mentioned, there are a number of examples of TSA development in the literature. Chandrasekaran is responsible for the concept of the *generic task*, a set of TSA's each applicable or generic to domain requiring the particular reasoning strategy. Steele (Steele, 1990) has also developed a methodology for identifying the generic tasks in complex problems, and providing a framework for developing such systems. Others have examined TSA's as well, though perhaps not as a direct research thrust. For example, the SOAR system of Newell allows for the development of TSA's within its architecture, and the BB1 architecture of Hayes-Roth has been used to develop a number of TSA's including 3-D arrangement (Hayes-Roth et al., 1986; Garvey et al., 1987).

The promise of such task-specific architectures is that once a "sufficient set" of them have been identified, complex problem-solving can arise from a composition of appropriate methods. What must be shown is how this promise of integration can be accomplished smoothly and in a generic way. That is to say, what kind of architecture can support use of many different task-specific architectures working together to solve some overall goal or set of goals?

2 TIPS (Task Integrated Problem Solver)

TIPS was an experiment in designing an architecture that would support generic integration of potentially heterogenous agents. In particular, it's goal was to investigate how available TSA's (hierarchical classification, abductive assembly, routine design, etc.) could be used to solve larger, more difficult problems. As an experiment it was not intended as a general-purpose architecture for problem-solving, but rather as a testbed in which certain goals could be examined in the context integrating generic task methodologies. These goals included:

1. The process of integration should be dynamic, that is the problem-solver sequence should not be preset. Instead, the selection of which problem-solver to apply next should depend on contextual information such as: the problem state, previous problem-solving history, knowledge available and other factors. In other words, different sequences of problem-solving should result if different information/data is provided.

2. The integration process should be based the goal-structure of the overall problem. This structure can be used to determine which goal is "current" i.e., which goal is most attainable/important/appropriate given the present problem-solving context. The architecture, using this information, then maps the most "appropriate" problem-solver to achieving the current goal.

3. The problem-solvers used in the integration architecture could be heterogeneous. Each problem-solver need only describe which goals it can achieve and what information, broadly speaking, it needs to achieve that goal. The result is a more modular system where the roles are clearly indicated in the architecture but the actual implementation of the problem-solver is not. As long as the problem-solver can fulfill its role and can communicate its' results then it can be used in the architecture.

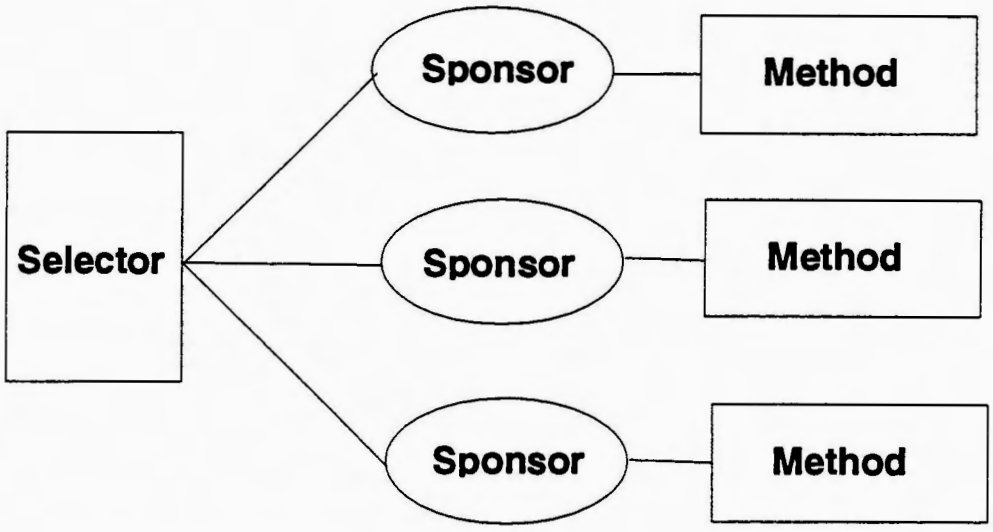


Figure 1: The structure of a sponsor-selector system.

4. The actual selection process of which problem-solver to invoke next requires knowledge not only of what goals it can achieve, but what *particular* knowledge a problem-solver has for the present problem. That is, knowing how to rebuild carburetors does not necessarily imply that you can rebuild a Model-T ford or a Testarosa carbuertor, or at the very least your confidence in that decision might vary based on particulars. Thus selection is based on both general capabilities and particular knowledge of a problem-solver.

5. The resulting system should provide a high level of explanation of why a particular problem-solver was invoked. This is important from the viewpoint of the knowledge engineer who has to use the system and modify it to meet changing needs.

We searched for a control architecture that would support these goals while avoiding other committments on problem-solving approache that would interfer with heterogeneity of the underlying methods. Described in detail elsewhere[1], the basis for this representation is the Sponsor-Selector system first used in DSPL (Design Specialists and Plans Language) (Brown and Chandrasekaran, 1989). As used in DSPL, a sponsor-selector system is used to select from a set of plans which is most "appropriate" for solving a design/planning problem. As used in TIPS, the role of the sponsor-selector has been broadened to control decisions, i.e., which problem-solver is most appropriate for solving the current goal.

A sponsor-selector system, as shown in Figure 1, consists of a hierarchy of three parts: at the top a *selector*, under the selector some number of *sponsors*, and under each sponsor a *problem-solver invocation*. In short, the available problem-solvers are grouped under the selector as sponsor-method pairs, where each sponsor provides appropriateness measures for its associated problem-solver invocation. At any control choice point, i.e., some point in the flow of problem-solving at which another problem-solver could be invoked the overall control

[1]See (Punch, 1989; Punch and Chandrasekaran, 1990) for more details.

process is to run all the sponsors to rate their associated invocations, then have the selector choose the next problem-solver to be executed based on the sponsor values and other data.

The sponsors are therefore used as "local" measures of how appropriate a problem-solver is for achieving the current goal, while the selector takes a more "global" view of selecting which of the available problem-solver is the "best" under the present circumstances. While a number of appropriate knowledge is possible, Both the sponsors and selectors typically encode their knowledge in a patter-match table that indicates what features are important for making the decision and how those combinations of features contribute to the final answer (selection or appropriateness measure).

One can think of the process of coding a TIPS problem-solver in terms of a craftsman's approach to building some artifact. The first step is an analysis of the overall problem, mostly in terms of already understood approaches. For the carpenter, does the artifact need lathing, cutting, sanding, routing etc., each a high-level solution to particular kinds of problems. Having identified the *task structure* of the problem, the process is to begin solving each sub-problem (meet the goal of each subproblem) by selecting which tools to use, in conjunction with other problems and their constraints. Thus TIPS is a *routine problem-solver* in the sense of solving known problems with tools whose capabilities/effects are understood.

2.1 Sponsors

Each sponsor contains information about when its associated problem-solver is appropriate. The sponsor evaluates this information and yields a discrete value indicating how appropriate the problem-solver is. The sponsor's knowledge is represented as a pattern match group similar to the knowledge groups of CSRL (Bylander and Mittal., 1986), the tool used to build hierarchical classification systems. This representation is a table in which each column, except the last one, is a particular query about the problem-solving state of the system. Each row is a combination of answers to each question, a pattern of response. If a row "matches", that is, each pattern element of the row is true for the query with which it is associated, then the last element of the row is the symbolic value for that pattern match group. Thus the last column is a list of the symbolic values that could be potentially returned by the pattern match group. Control of row examination is of the simple "first true" type; that is the rows are evaluated in order until one of them matches. Thus the representation allows a knowledge engineer to investigate knowledge patterns in a specific order. Possible patterns of response are ruled out as the rows are evaluated until a matching pattern is found. If no patterns match and all have been examined, then some default value is returned, usually "unknown".

Consider the example pattern match group of Figure 2 for the classification hierarchy sponsor. There are two queries: Query 1 represented by Column 1 is, "Has hierarchical classification been applied yet?" , and Query 2 is, "Is the present diagnostic explanation complete?" (that is, explains all the data). Row 1 represents the pattern of response:

> If hierarchical classification has not yet reached completion, represented by the F or false answer to Query 1, then regardless of whether the explanation is complete, represented by the ? or "don't care" answer to Query 2, then return a 3.

1) Has the classification method been applied yet?
2) Is the present diagnostic explanation complete?

Query 1	Query 2	Plausibility Value
F	?	3
T	F	1
T	T	-3

Figure 2: An example match group which determines the classification sponsor appropriateness value

The return values can be any discrete range of symbolic values to represent the appropriateness scale; in this case it is one of -3 (for totally inappropriate) to +3 (for totally appropriate).

Note that the pattern information represented in the rows can be more complicated expressions to match the kind of information returned by the queries, not just simple true/false distinctions.

There are two kinds of information available as appropriateness pattern match features:

1. Information about what problem-solvers have been applied so far. These are questions about which methods have run so far, if at all, and when they ran. The architecture provides automatic access to these kinds of questions, such as: "Did the hierarchy run last?", "Has the abductor run in the last 3 cycles?", etc. Query 1 is an example of a query for this type of information.

2. Information concerning goal achievement, such as: "Is the explanation complete?" or, "Has this finding been explained?". This kind of information has to be supplied by the programmer who assembles both the behaviors and the pattern match knowledge and is not part of TIPS. Query 2 is an example of a query for this type of information.

2.2 Selectors

Each group of sponsors and their problem-solver invocations are grouped under a selector (see Figure 1). The selector does two things:

1. It organizes the set of problem-solvers that are available to be run on each cycle of evaluation.

2. It selects which problem-solvers is next invoked based on the appropriateness ratings of the sponsors and other knowledge.

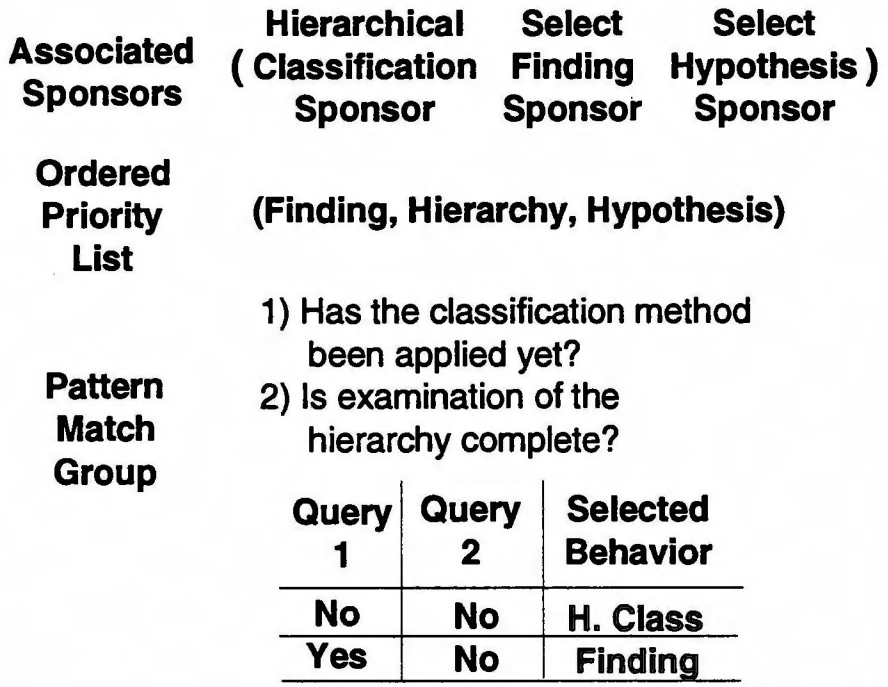


Figure 3: The internals of a selector

If a selector is activated, only the problem-solvers under it are available for evaluation on each selection cycle. Other groups of sponsor-selectors may be activated as a result of some invocation being evaluated, but only one sponsor-selector system at a time is responsible for selecting problem-solvers to evaluate.

The main criterion selection are the appropriateness measures returned from its sponsors, but often this is not enough. The sponsor pattern match information *should* be coded with only local considerations in mind. In other words, it should not be the knowledge engineer's job to have to program each sponsor such that only one should be appropriate under all circumstances. Ideally, each sponsor is coded with little consideration of other sponsors with more global considerations of proper selection left to the selector.

The selector knowledge can be encoded in three ways, as shown in Figure 3.

1. The selection can occur simply based on appropriateness measures. If a clear selection is available, i.e., there is exactly one highly rated problem-solver, then it is invoked. If no clear selection is available and no other selection knowledge is available, then a random choice from among the best candidates is selected, i.e., those with the highest plausibility rating.

2. If there is a tie among the highest rated problem-solvers, tie breaking knowledge in the form of an *ordered priority list* can be used to break the tie. That is, the knowledge engineer can provide knowledge of problem-solver priority which determines which is selected when appropriateness ratings are not enough. If more than one sponsor is

marked as appropriate, then the first appropriate problem-solver in the priority list will be chosen. As shown in Figure 3, if the hierarchy and the finding-selection sponsors are tied, then findings-selection will be chosen.

3. For special situations, it is possible to override the normal choice mechanism with some pattern match knowledge similar to that found in the sponsors. If the pattern is matched, then the choice indicated in the matching row is used. If no match occurs, then the default priority list choice is used. The advantage of this is the processing time that can be saved to make a specific decision that is already determined by problem-solving analysis. The disadvantage is its inflexibility and narrow applicability. In the figure, if hierarchical classification has never been applied, as opposed to applied but not yet completed, then it should be the selected method. Notice this overrides the priority list.

The overall programming of the TIPS architecture is split into two parts:

1. Local decisions about appropriateness are coded in the sponsor in terms of knowledge about problem-solving progress and history of module activation, i. e. , both success/failure information and information concerning module selection.

2. Global decisions that concern actual choice of module invocation (based on sponsor results and other information) are coded in the selector.

As stated previously, multiple groups of sponsor-selectors can and often do exist, but only one such system is active at a time. Others may be activated by a selector, but that one module-activated selector will remain the only active system until it completes.

Completion of cycling through a sponsor-selector systems is accomplished by a *Return* or *Fail* sponsor. These sponsors do not have any associated modules. Rather, they indicate when that particular sponsor-selector has finished its work or when it has failed to accomplish the task for which it was activated, that is whether the goal associated with the selector has been reached or not. These two sponsors often are at the highest priority since failure or completion are the two exit conditions. Setting of the Return and Failure sponsors are programmed just as any sponsor, by the knowledge engineer.

3 The Med-TIPS example

Having created the architecture, the we now needed to investigate its usefulness in the context of a complex domain. We choose a medical diagnosis problem which would include many different aspects of problem-solving. The domain of liver and blood disorders (Punch and Chandrasekaran, 1990). The following briefly summarizes the modules used in the system as shown in Figure 4 (see (Punch, 1989) for more details). In particular it shows two things. First, it describes the goals of the system and the usefulness of each problem-solver for meeting those goals. Second, it summarizes the *appropriatness* knowledge for each problem-solver as encoded in the sponsors used in the system.

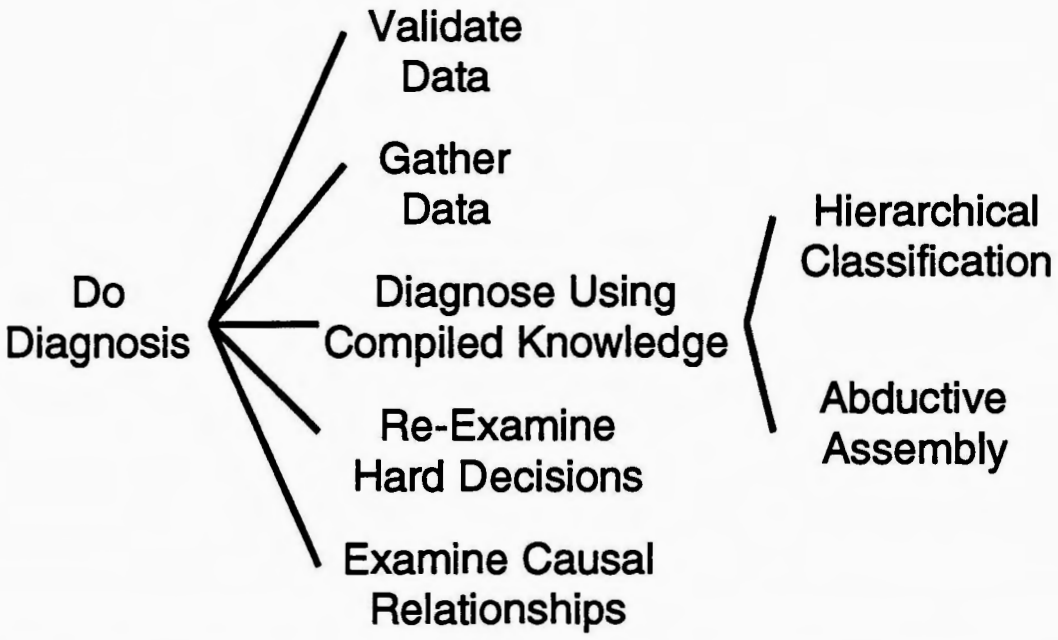


Figure 4: The task structure of the medical diagnosis system implemented in TIPS .

- Compiled knowledge diagnostician. This problem-solver is responsible for creating diagnostic explanations using the abductive-assembly/hierarchical-classification architecture found in RED (Smith et al., 1985; Josephson et al., 1987). It is used to meet the Diagnose using Compiled Knowledge goal.

  Briefly, this problem-solver has two subgoals, Hierarchical Classification and Abductive Assembly. Hierarchical Classification (Bylander and Mittal., 1986) is a problem solver that examines a hierarchy of (in this case) malfunction categories to determine the plausibility of malfunctions given the present circumstances. This plausibility information is used by the next phase, Abductive Assembly (Josephson et al., 1987), to assemble a subset of the malfunction hypotheses such that the resulting *explanation* is most plausible, consistent and covers the findings.

  When Appropriate: This problem-solver is appropriate: as the very first problem-solver in a run (to create an initial diagnosis), if more data has been gathered, if a datum value was shown to be invalid and replaced by a validation method, or if causal reasoning has suggested an new malfunction hypothesis to consider for use in the composite explanation.

- Data gathering problem-solver. This problem-solver gathers evidence that can establish or rule-out pertinent diagnostic hypotheses. It is used to meet the Gather Data goal.

  In this particular case, the data that is needed next is determined by the nodes in the hierarchical classifier that suspended due to lack of data. If this data can be made available, then further exploration of the hierarchy is enabled.

When Appropriate: This problem-solver is appropriate when: the hierarchical classifier cannot explore leaf level hypotheses due to lack of data, or one of the potential hypotheses in a hard decision (see item "Redoing hard decisions" in this section) has an unanswered question in its specialist.

- Data validation problem-solver. This problem-solver checks the validity of certain data that have been questioned during the run of the compiled knowledge diagnostician problem-solver. It is used to meet the Validate Data goal.

  Described in some detail elsewhere (see (Chandrasekaran and Punch, 1987)), the idea is the following. Rather than rely strictly on statistical "averaging" based on multiple sensor readings for one datum, this problem-solver uses expectations derived from partial hypotheses already formed by diagnosis. If these expectations are not met (i.e. a partial conclusion of "Liver Disease" requires some change in liver enzyme values and those values report in presently as normal) then there is a potential data problem. These unmet expectations are flagged and further investigated by specific test procedures that validate their value.

  When Appropriate: This problem-solver is appropriate if, in the process of diagnosis, a plausible composite explanation is generated whose data expectations are not met. Those data that do not meet expectation are noted as questionable and are submitted for validation.

- Redoing hard decisions problem-solver. This problem-solvers examines the assembly of alternate diagnostic explanations stemming from a hard decisions in the compiled knowledge diagnostician problem-solver. It is used to meet the Change Hard Decisions goal.

  A hard decision is a situation reached in abductive assembly when no clear criteria is available to differentiate among a set of hypotheses that offer to explain a finding. For example, the hypotheses A, B and C all offer to explain finding F and all have the same plausibility. If the only criteria available for differentiating which is the "best" hypothesis to explain F is plausibility, then there is no basis for a choice and a hard decision occurs. Typically the system makes a random choice and goes on but it also records the system state at that point so that the user may later go back and explore other possible solution paths.

  When Appropriate: This problem-solver is appropriate when a hard decision has resulted during the abductive assembly process.

- Causal reasoning problem-solver. This problem-solver applies the coherence view of causal modeling of (Punch, 1992) to the diagnostic explanations generated by the compiled knowledge diagnostician problem-solver. It is used to meet the Do Causal Reasoning goal.

  Briefly, associated with each diagnostic hypothesis is some generic knowledge about how the malfunction it represents modifies normal function. These changes are broadly categorized as functional changes (Anemia causes loss of oxygenation function), connective changes (blocked Common Bile duct causes back-up of bile) and structural

changes (swelling of pancreas can push on the liver). If a possible link between the elements of a partial abductive assembly hypothesis are found, their relationship is explored using a model of the involved elements to determine some causal results not yet available to the compiled diagnosis component.

Consider the example (used in the Med-TIPS system) of the interaction between hemolytic anemia and an as yet non-specific liver disease. If hemolytic anemia causes a functional increase in iron products in the blood and the liver is responsible for clearing these products, a relationship between hemolytic anemia and the as-of-yet non-specific liver disease can be explored to discern a possible disease process. This can be done by simulating the liver model of "iron-clearing" with an additional parameter of excess iron. The results of this simulation may lead to a better understanding of the liver disease and suggest tests to validate the conclusion.

When Appropriate: The causal reasoning problem-solver is appropriate when a possible causal interaction has been noted between hypotheses of the present explanation.

3.1 An example Case

Having described the problem-solvers, goals and control knowledge of the system, we can now describe in abbreviated detail an example run of the system on a typical problem. Many details have been left out but the trace is included to give the reader a feel for the type of problem-solving that goes on in the system. It is meant to only show how the control flow is determined by the interaction of the sponsors of the methods and the goals of the problem. Note that depending on the initial data, the system may solve the problem in a completely different way.

The case is taken from a clinicopathological conference at the Albany Medical College (B-56469), May 6, 1954 and cited in Harvey and Bordley (Harvey and Bordley, 1972),

Step 1: Given the data base values and the findings to explain the hierarchical classifier/abductive assembler, which always runs first, comes up with an initial diagnosis of (`ThalassemiaMinor Hepatomegaly`).

Step 2: The explanation generated is rated via two criteria to determine if it is "good enough". The criteria in this case are that the explanation explain all the findings and that it consist of hypotheses that are most detailed, i.e., come from the leaf level of the hierarchy. This explanation is considered not good enough (the hypothesis elements explained all the findings but were not all at the leaf level of the hierarchy, i. e. , not an explanation of the greatest detail possible), so the sponsor-selector system is invoked. Most method selection will not occur through the action of the sponsor-selector mechanism.

Step 3: The sponsors of the TIPS architecture were evaluated The ones deemed appropriate this time were Gather More Data and Causal Reasoning, both of which were rated +3 on a scale of +3 (completely appropriate) to -3 (completely inappropriate). Gather More Data was appropriate since the hierarchy was not explored to leaf level. Causal Reasoning was appropriate because a functional relationship was noted between ThalassemiaMinor and Hepatomegaly. The relationship is based on the production of iron by ThalassemiaMinor and the normal liver operation of consuming iron associated with Hepatomegaly. The Causal

Reasoning problem-solver was selected based on the tie-breaking knowledge of the sponsor's problem-solver priority list since causal reasoning has a higher priority than gathering data

Step 4: The causal reasoning problem-solver focuses on the functional relationship of iron, based on `excess-produce-iron` of `ThalassemiaMinor` and the `consume-iron` of `Hepatomegaly`. Note that the latter is a normal function of liver for which there is no evidence of change.

Step 5: Based on the proposed interaction of iron, a search is made for a model of iron-metabolism of liver. It is found in the Liver specialist associated with "iron-metabolism". The liver's iron-metabolism function is simulated on the initial conditions of excess iron, derived from the excess production of iron from `Thalassemia Minor` stored with its interaction knowledge. Note that it also searches for an effect of `consume-iron` on Thalassemia Minor, but finds none. This is important since the interaction could occur in either direction.

Step 6: The simulation yields the state of `iron-deposits-in-liver-cells`, which is noted as abnormal by the data base (the data base stores normal ranges for all data). The causal problem-solver searches to see if `iron-deposits-in-liver-cells` is associated with any known disease. It does so by searching another association list of states-to-malfunctions stored in the Liver specialist. An association is discovered between `iron-deposits-in-liver-cells` and the disease `hemochromatosis`.

Step 7: The causal model runs the compiled knowledge of the hierarchical classifier associated with the disease `hemochromatosis` to determine if there is enough evidence to establish it. Note that this is an example of method invocation bypassing the sponsor-selector system. Such bypassing is appropriate here since the mechanism requires that such potential solutions be confirmed. However, it could also have been programmed through the normal sponsor-selection mechanism with no change in functionality. In any event, the node establishes so the disease `Hepatomegaly` is replaced as a hypothesis for the abductive assembler by `hemochromatosis` since the latter is a particular type of the former and the abductive assembler is trying to create the most detailed explanation possible. The causal reasoning problem-solver then returns control to the sponsor-selector.

Step 8: The explanation has not yet changed, the diagnostic explanation is still not "good enough" so the TIPS diagnostic architecture is again invoked. This time, and the Compiled Knowledge Diagnostician Problem-Solver and the Gather Data Problem-Solver are the only ones appropriate, both rated at +3. The Compiled Knowledge Diagnostician was appropriate because Causal Reasoning suggested hemochromatosis as a hypothesis for use in the composite explanation. The Gather Data Problem-Solver is appropriate since the hierarchy was still not explored to leaf level. Since the Compiled Knowledge Diagnostician Problem-Solver has a higher priority, it is invoked next.

Step 9: The Compiled Knowledge Diagnostician Problem-Solver creates an alternate explanation with the elements (`ThalassemiaMinor hemochromatosis`). That explanation is complete and detailed, so diagnosis is finished and the case ends.

4 Problems with TIPS

While TIPS has formed the basis for our exploration of task-specific integration, it of course had a number of flaws. The first problem is one of goal representation. Figure 1 is a di-

rect representation of the problem-solvers used to achieve goals in the composite design problem, but not the goals implicit in guiding the problem-solving. In the current TIPS implementation, the problem-solvers are directly represented, but the goals are only represented implicitly in the sponsor-selector system.

The second, perhaps more important, problem is the encoding of the sponsor-selector knowledge and the problem-solver interaction knowledge. This is really the problem of living up to the goal of the heterogeneity goal we spoke of earlier. In order for the control-selection process to proceed independent of the problem-solving methods used, two issues must be addressed. First, we must discover a way to encode the *ontology* of what the problem-solver requires to solve the problem. This boils down to understanding the terms and their meanings used by the problem-solving method. For example, Gruber (Gruber, 1992) describes the language Ontolingua which allows a knowledge engineer to represent the committments a problem-solving method makes in terms of representations.

Second, it is clear in retrospect that the Med-TIPS system is important not so much for the TIPS architecture it uses, but for the *appropriateness* knowledge it requires in order to cement the various methods together. It is our opinion at this point that the real problem to be further investigated in integration is to describe what a problem-solver is "good for" in a generic way just like Ontolingua provides a generic way to describe a problem-solver's ontology.

5 A Formal Approach

Our most recent work has reconstructed our approach to task-specific integration, using our experience with TIPS , by more formally describing an approach to integration.

We define a task to be a 3-tuple:

$$T = (G, PS, S)$$

where G is a set of goals, PS is a set of problem-solvers to be used in achieving the goals and S is a selector for deciding which problem-solver to activiate on the present problem solving cycle. We further refine the definition of G and PS as follows:

$$G = (Gset, \sigma(g, Pstate))$$

$$PS = (PSset, \delta(ps, GSet^*, Pstate))$$

$GSet$ is the goal set itself and $\sigma(g, Pstate)$ is a function that maps $g \in GSet$ and the present problem state into $v \in V_g$, where V_g is a vector that lists the range of "activiation" of goal g under the present problem state. In other words, σ indicates whether the goal g should be pursued during the present problem-solving cycle and the degree of confidence in that decision on a scale given by V_g. The character of V_g is determined by the application, it could be as simple as the set (T, F) or real-valued such as $0 \leq v \leq 1$.

$PSset$ is the set of problem-solvers themselsves and $\delta(ps, GSet^*, Pstate)$ is a function that maps $ps \in PSset$, $GSet^*$ which is $GSet$ with activity values set by σ and the present problem state into $v \in V_ps$, where V_ps is a vector that lists the range of "applicability" of

problem-solver *ps* for achieving goals identified as active by σ. That is, given a set of goals and how actively they should be pursued, δ indicates how applicable each *ps* is for achieving those goals and the confidence in that decision on a scale given by $V_p s$. As with the goals, the character of $V_p s$ is determined by the application.

The above definition of task has a number of interesting characteristics:

1. The goals of the task and the structure of those goals (if any beyond that of a set) are separated from the decision making process of which goal(s) is active and to what degree. Thus a task experimenter is free to explore alternate goal sets/structures with the same activiation process and vice versa.

2. The problem-solvers which are used to realize the task are separated from the selection control procesa. Again, the task experimenter is free to explore alternate problem-solvers for achieving the same goal and vice versa.

3. The result of 2. above is that the form of the problem-solver need not be known at least at the level of deciding how appropriate the problem-solver is for achieving a goal. This allows for the integration of disparate or heterogeneous problem-solvers

   Note however that a useful characteristic between the δ function and a problem-solver *ps* is that δ be able to query *ps* as to how appropriate *ps* perceives itself to be in terms of available knowledge. That is, δ may know generally what kind of goals *ps* can achieve (what kind of problems it can address) but only *ps* may know whether it has the *particular* knowledge necessary to solve the present problem.

4. The result of 3. above is that the characteristic of problem-solving is dependent on the kinds of problem-solvers and goals the task addresses. For example, a uniform problem-solving architecture such as SOAR can be accomidated by making the problem-solvers adhere to the problem-space and subgoal approach. Alternately, a society of mind or pandemonium architecture is possible where quite different kinds of problem-solvers are "competing" for invocation rights and some kind of negotiation must ensue between the selector S and the δ functions of each.

5. A Task/Subtask structure can be defined explicitly by allowing a Task problem-solver invocation to recursively call a subtask of the same three-tuple structure (potentially with a different set of goals, problem-solvers, δ and σ). Thus a problem-solver can either be "atomic" or a call to invoke a subtask. While the invocation of task, subtask and problem-solver may not be known until runtime, the choices are explicit. If such structure is not desired, all problem-solvers can be atomic and located at the same top-task level.

6. Both the number of active goals and invoked problem-solvers is not restricted by the architecture. In other words, multiple goals could be pursued in parallel by invoking multiple problem-solvers. However, this introduces potential problems in the "parallel" update of information which must be mediated at the task level.

6 Future Research

The above formal specification on the integration environment provides our next testbed in which we can further investigate our ideas of task/problem-solver integration. In particular it addresses the conflation of problem goals and the problem-solvers used to achieve them.

However, we are only now beginning to investigate the formalization of the *delta* function, that is determining the "what a problem-solver is good for" concept in a language that can be used for heterogeneous problem-solvers.

Our original work with the TIPS system and dynamic integration of task-specific problem-solvers has been positive. The instantiation of the Med-TIPS system gave us a testbed for investigating the effects of dynamic integration in a broad problem area. In particular it pointed to the more important problems of goal separation and problem-solver appropriateness representation.

References

Brown, D. and Chandrasekaran, B. (1989). *Design Problem Solving: Knowledge Structures and Control Strategies.* Pitman, London, UK.

Bylander, T. C. and Mittal., S. (1986). CSRL: A language for classificatory problem solving and uncertainty handling. *AI Magazine*, 7.

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. (1987). Towards a functional architecture for intelligence based on generic information processing tasks. In *Proceedings of the International Joint Conference on Artificial Intelligence 87*, pages 1183–1191. International Joint Conference on Artificial Intelligence.

Chandrasekaran, B. and Punch, W. F. (1987). Data validation during diagnosis, a step beyond traditional sensor validation. In *AAAI87*, pages 778–782.

Garvey, A., Cornelius, C., and Hayes-Roth, B. (1987). Computational costs versus benefits of control reasoning. In *AAAI87*, volume 2, pages 110–115. Morgan Kaufmann.

Gruber, T. (1992). Ontolingua: A mechanism to support portable ontologies. Technical report, Stanford University. Version 3.0 report.

Harvey, A. M. and Bordley, J.. (1972). *Illustrative Case I in Liver Diseases*, pages 294–298. W. B. Saunders.

Hayes-Roth, B., Buchanan, B., Lichtarge, O., Hewett, M., Altman, R., Brinkley, J., Cornelius, C., Duncan, B., and Jardtzky, O. (1986). PROTEAN: Deriving protein structure from constraints. In *AAAI86*, volume 2, pages 904–909. Morgan Kaufmann.

Punch, W. F. (1989). *A Diagnosis System Using a Task Integrated Problem Solving Architecture (TIPS), Including Causal Reasoning.* PhD thesis, The Ohio State University.

Punch, W. F. (1991). TIPS (task-integrated problem solver), a task-specific integration architecture for heterogeneous agents. In *Proceedings of the AAAI-91 Workshop on Cooperation Among Heterogeneous Intelligent Systems*, pages 1–10, Anahiem, CA.

Punch, W. F.. (1992). Interactions of compiled and causal reasoning in diagnosis. *IEEE Expert*, 1(7):28–35.

Josephson, J. R., Chandrasekaran, B., Smith, J. R., and Tanner, M. C. (1987). A mechanisim for forming composite explanatory hypotheses. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-17(3):445–454.

Punch, W. F. and Chandrasekaran, B. (1990). An investigation of the roles of problem-solving methods in diagnosis. In *Proceedings of Second Generation Expert System's Conference*, pages 25–36.

Smith, J. W., Svirbely, J. R., Evans, C. A., Strohm, P., Josephson, J. R., and Tanner, M. C. (1985). Red: A red-cell antibody identification expert module. *Journal of Medical Systems*, 9, Issue 3,:121–138.

Steele, L. (1990). Components of expertise. *AI Magazine*, 11(2).

THE VALUE OF DESIGN KNOWLEDGE

Yoram Reich
Engineering Design Research Center
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213, USA

ABSTRACT

The quality of design knowledge a system has substantially influences its performance; good quality design knowledge is a true asset. Often, the terms knowledge, its quality, and how it is measured, are left vague enough to accommodate several interpretations. This paper articulates two definitions of knowledge and their associated measures. A detailed example of knowledge evaluation using the measures is described. The example demonstrates the value of knowledge quality evaluation. This value is an addition to the methodological function that evaluation provides. Finally, the paper briefly discusses the scope of the measures and their relationships.

1 INTRODUCTION

Design knowledge exists in many ways and qualities: It may be referred to as being accurate, perfect, plausible, approximate, cheap, expensive, implicit, explicit, opaque, etc. Sources of good quality design knowledge are limited; they mostly exist in the minds of expert designers, while a small portion is documented in the literature. In order to identify which knowledge is good, one must have the ability to evaluate knowledge quality. The topic of knowledge evaluation raises several prerequisite issues. First, what do we mean by design knowledge? Second, how is something in general being evaluated or measured? This question may seem unimportant, but in fact, it is central to the discussion since it deals with the concepts of *measure* and *value*.

A measure of design knowledge quality can serve several functions. The first and most general function is practical: The identification of good design knowledge for use by designers or inclusion in computer systems. The second function is methodological: The evaluatation of prototype design systems developed in research and determination of their relative merit. This evaluation is essential for providing feedback on research progress and supports the refinement of ideas (Cohen and Howe, 1989). When evaluating design systems that learn, this function becomes mandatory.[1] A third function is manifest when complex design systems with several modules that solve similar problem compete on resources. In this situation, knowledge quality measures can identify which module to invoke. In this study, we concentrate on the second function: the evaluation of knowledge embedded in design support systems.

The subject of this study belongs to a larger set of methodological and practical issues of evaluation. As we see later, the evaluation of knowledge quality is tightly coupled with the evaluation of

[1] In fact, research on learning provides significant insight on the measurement of knowledge since the evaluation of learning progress requires the evaluation of knowledge acquired.

intelligent systems. The state of evaluating such systems was documented by Green and Keyes (1987) . They described the reluctance of system developers to verify and validate their products. The lack of effort spent on evaluating software systems in general, and expert systems specifically is also discussed by Adelman (1991) and Cohen and Howe (1989). The latter also demonstrated the benefits from evaluation to research and the necessary steps needed to allow evaluation to be performed.

If evaluation of systems is addressed in studies, it is often performed in an *ad hoc* manner. To illustrate, Zuse and Bollmann (1987) discussed the chaotic state of measures in software engineering. Most of the software measures appearing in the literature do not conform to the notions of measurement theory (Stevens, 1946). The argument is not that one measure is better than the other, but that a true measure must have certain properties. First, there must be two systems, *observed* and *formal*, where a system is a mathematical entity consisting of a set of objects A, relations on A, and a binary operation on A. For example, an observed system may contain the set of physical objects, the relation "heavier-than," and the operation of assembling two objects. The formal system can consist of the set of positive real numbers, the relation "larger-than," and the addition operator. Second, there must be a function from the observed to the formal system that preserves the relations and operation, called *homomorphism.* In the above example, a mapping that assigns each object its weight is a homomorphism. This allows a meaningful assignment of *value* to the weight of physical objects. Most measures of software systems do not have these properties, although all *should have* according to Zuse and Bollmann. Similarly, measures that evaluate knowledge should have these properties.

This paper proposes four techniques for evaluating knowledge: structural qualitative and quantitative, and functional qualitative and quantitative. It contrasts their advantages, disadvantages, and appropriateness for different purposes. The paper illustrates them in the evaluation of an experimental design system. It does not deal with the debate whether machines can have knowledge or be intelligent (Searle, 1980; Dreyfus, 1979); and whether knowledge in a system can be "objectively" evaluated in any way.[2] More importantly, the paper (but not the author) assumes the empiricist or analytical paradigm of philosophy which favor the above ideas on measures. It attempts to raise the issues and focus the attention on the methodological and practical implications of the evaluation of knowledge within the empiricist paradigm, thereby motivating further research in this direction. Such research may extend the use of constructs from continetal philosophy in the evaluation of knowledge, hence, address some of the deficiencies in knowledge evaluation briefly pointed later.

The remainder of this paper is organized as follows. Section 2 provides two definitions of knowledge and briefly discusses them in the context of design. Section 3 describes four evaluation measures of knowledge. Section 4 describes the design system BRIDGER that is used to demonstrate the evaluation measures. Section 5 provides a detailed example of measuring the bridge design knowledge that BRIDGER has. The extent of the evaluation allows the appreciation of the many concerns affecting the selection and application of different evaluation measures of knowledge. Section 6 concludes the paper.

[2]This comment disagrees with Gaines and Shaw (1989) reference to "*objective knowledge*" as the knowledge arrived by consensus among experts; such knowledge is subjectively created within a constructivist viewpoint. Furthermore, although most empiricist will claim that a measure as described before can be objective, they ignore the fact that the mapping from the observed to the formal system is theory-laden and subjective.

2 WHAT IS (DESIGN) KNOWLEDGE?

In defining knowledge it must be understood that the definition dictates the type of evaluation measure that can be applied or that may best apply. Many studies on knowledge-based systems avoid the question of what knowledge is by discussing knowledge representation. Implicitly, these studies define knowledge as *whatever is represented*, henceforth termed the *structural definition*. Knowledge is therefore a static entity; it may include facts, axioms, derivations, causal relations, mathematical models, etc.

The structural definition is in contrast to the *functional definition* which states that knowledge has a purpose and it is defined as *what a system has that allows it to attain goals*. Knowledge cannot be observed directly, but indirectly through observing the "intelligent" behavior of a system.

Doyle (1988) termed these definitions as: (1) *explicit knowledge* which is what is represented; and (2) *implicit knowledge* which is what can be deduced from explicit knowledge. Doyle argued that adopting the explicit definition has several limitations: (1) it cannot explain actions that are not logical, default or nonmonotonic reasoning; (2) it cannot explain some psychological phenomena; and (3) it cannot handle inconsistencies that naturally arise, for example, in knowledge generated from several experts.

2.1 Structural Definition

The structural definition has several appealing properties. The main one is knowledge sharing. If knowledge is what is represented, then knowledge can be abstracted from the system using it and shared with another reasoning mechanism. Two "quantities" of knowledge could be added to yield a larger knowledge base, or knowledge could be transferred between knowledge systems (Neches et al., 1991). A knowledge interface format (KIF) is developed to facilitate this transfer. Another appealing property of the structural definition is that it facilitates easy evaluation performed by simply inspecting the declarative structure of knowledge. This is much cheaper than executing behavior assessment experiments.

The structural definition detaches knowledge from its method of acquisition and use. This may cause difficulties. For example, in order for systems with different uncertainty management mechanism to cooperate they must establish an interface between them where uncertainty values are translated via a fractionally linear function (Kreinovich and Kumar, 1991). If translation is not performed, the ability to use the knowledge decreases (e.g., the decrease of diagnosis performance in MYCIN when the method of probability calculation is changed (Shortliffe, 1976)). This suggests that one cannot generate and "plug" knowledge that was derived by some mechanism into an expert system and expect it to function properly.

Following the structural definition, analytical philosophers usually place restrictions on what they accept as knowledge (Fetzer, 1990). Knowledge is a set of beliefs that are consistent; a belief that can be derived from the other beliefs is also part of the knowledge; and if there is evidence that a belief is warranted, then it should be part of the knowledge, provided that all the other evidences also warrant this.

2.2 Functional Definition

Newell (1982) opposed the structural view of knowledge. He argued that "*knowledge* is a distinct notion, with its own part to play in the nature of intelligence," independent of representation. In order to define knowledge, Newell defines the "principle of rationality: If an agent has knowledge that one of its actions will lead to one of its goals, then the agent will select that action." This principle governs the use of knowledge for making the appropriate actions. Knowledge is therefore defined as "whatever can be ascribed to an agent, such that its behavior can be computed according to the principle of rationality. ... Knowledge is a competence-like notion, being a potential for generating action;"[3] Therefore, knowledge is evaluated functionally. The relation of knowledge and representation is clear. First, "representations exist at the symbol level." Second, "knowledge serves as the specification of what a symbol structure should be able to do."[4]

2.3 Design Knowledge

Design knowledge can be defined as knowledge in general, except that it is about design. Since the nature of design processes and knowledge is orthogonal to the question of its evaluation, we will refer to design as a process mapping from requirements or needs to a description of an artifact. In the structural definition, design knowledge may include facts about physical sciences; facts about modeling techniques of, and experiental heuristics about, artifacts; and information about how artifacts may be structured, how do they behave, and how they may be built. At this level, knowledge may be abstracted from its representation; there need not be any commitment to specific representational formalism.

In the functional definition, design knowledge is what can be ascribed to a design agent, such that its design behavior can be computed according to the principle of rationality. This means that knowledge can be described in terms of its operation to satisfy the goals of the design system. If a system cannot use a piece of information in its reasoning, then, the system *does not have* this knowledge.

We argue that the ultimate purpose of design knowledge is to act, reason, and create artifacts; design knowledge as an entity separated from use has no significant meaning and usefulness. We therefore favor the functional definition of design knowledge. Nevertheless, we do not, and probably cannot, resolve which definition of design knowledge is better or correct, we merely provide the definitions and contrast their evaluation.

Researchers on design have used both definitions and in various ways. Balkany, Birmingham, and Tommelein (1991) provided the description of several design systems at the knowledge-level. As such, they favor the functional view of knowledge being separated from implementation issues and geared toward functionality.

Bijl (1987) presented a view against conceptualizing the information within computers as knowledge—a statement indirectly in favor of the functional view (or even in line with continental philosophy).

[3] See Fetzer (1990, p. 127-130) for a critic on this definition. In addition to this criticism, continental philosophy has significantly different ideas about the nature of knowledge, its interpretation, and understanding (Mueller-Vollmer, 1985). See also (Mallery et al., 1986), for a review of hermeneutics related to computer understanding.

[4] The view of knowledge as a specification for its structural description was used, for example, by Levesque (1984) for building knowledge representation that can support certain functions; and by Kyburg (1988) too draw conclusions on how uncertainties should be represented to support decision making.

On the other hand, he described a system called MOLE as resting on the foundation of comprehensibility of design knowledge (p. 12). In MOLE, the internal representation is designed to be meaningful and to convey the functionality of the system—this corresponds to the structural view of design knowledge.

Brown and Spillance (1991) discussed knowledge compilation as a process that modifies the structure of design constraints. They view the quality of the modifications in the functional sense and evaluate them by performance tests.

Coyne, Rosenman, Radford, Balachandran, and Gero (1991) said that "in knowledge-based systems, the knowledge must be able to be made explicit in such a way that it can be inspected and understood independently of the way it is controlled." Therefore, they view knowledge in the structural sense.

Whereas researchers adopting the symbolic paradigm of design can subscribe to either definitions of knowledge, the studies employing neural networks in design must subscribe to the functional definition since the internal structure of a neural network does not reveal its knowledge content.

3 EVALUATION MEASURES

The evaluation of knowledge depends on its definition. Two measures, qualitative and quantitative, are proposed as supplying complementary information. Note that the qualitative measure should also lead to some quantification; otherwise, it will not be an appropriate measure or "objective" according to the empiricist paradigm.

There can be other measures of knowledge, for example, cost or simplicity. The cost of knowledge can measure the time and memory resources needed to manage it and the cost of using it to obtain performance. Simplicity can measure syntactic properties of knowledge structure or the trace of its execution. We view these and potentially other measures as secondary provided that they are not an exponential function of any of the problem characteristics.

3.1 Structural Measure

The process of *structural* evaluation is similar to a "brain surgery." Knowledge is measured based on its internal structure. Since a crucial aspect of knowledge is its use in performance tasks, the structural measure will probably involve *projecting* how the knowledge will perform while solving problems. This however will be hard without considering the mechanisms that manipulate that knowledge.

In the case of logic, knowledge can be structurally assessed according to its truth value. In rule-based systems (albeit without uncertainties) knowledge can be verified to be free of redundancy, conflicts, circularity, and incompleteness (Nazareth, 1989). This measure, however, is not useful for evaluating design knowledge which has all the above properties, but yet, used by humans effectively to generate designs. The qualitative and the quantitative variants of this measure are discussed next.

Qualitative measure. This measure can be traced to the assumption that humans can understand knowledge embedded in systems in a declarative form. It is based on past practice with small

systems having simple knowledge representation such as traditional production rules. However, in large and complex systems, it is hardly possible to understand what is the role of a small piece of knowledge and envision its potential run-time interactions with additional knowledge. Therefore, while the ability to inspect declarative knowledge can facilitate debugging, maintenance, and even education, it is difficult to support.

A different approach to support these concerns builds upon advanced mechanism that *manipulate* the knowledge, thereby transforming the evaluation from structural to functional. For example, GUIDON2 uses the knowledge of NEOMYCIN to teach students (Clancey, 1988) without letting the student look at the low level knowledge representation.

In machine learning, the comprehensibility of knowledge generated is described as an important measure by some researchers (Michalski, 1986). Note, however, that this measure has evolved from research on concept learning where the structure of knowledge is sufficiently simple and the task is a single-step classification that is easy to comprehend and evaluate. Furthermore, it is easy to predict the knowledge performance from its structure.

The execution of this measure requires domain knowledge. It is not clear how to quantify it or otherwise how to use it to compare between different knowledge contents.[5]

Quanitative measure. This measure quantifies the structure of knowledge. Viewing knowledge as information allows such quantification (Boulton and Wallace, 1973). A heuristic measure for assessing the quality of a classification, which is relevant to the example described later, is proposed by Gluck and Corter (1985). A generalized form of this measure, called *knowledge utility* (*KU*) is used in the example.

This measure is easy to apply, but requires a formalism that can be quantified. Compared to the qualitative structural measure, it is even less clear what is the relation between this measure and the design performance of a system.

3.2 Functional Measure

This measure evaluates knowledge based on its performance in various tasks. Adelman (1991) discussed three types of performance evaluation: (1) *experiments* which are suited for the early stages of system development and generate fully reproducible results; (2) *quasi experiments* which are for the operational stage of systems and consists of fully controlled artificial studies; and (3) *case studies* which are opportunistic and wholly unconstrained. The first two are quantitative measures and the latter a qualitative measure.

In the context of statistical decision theory, (and with several other assumptions,) one can quantify the value of knowledge. Skyrms (1990) defined knowledge as something that allows making *informed* decisions; he then used this definition to evaluate knowledge content. Such an evaluation is possible due to the simple nature of the knowledge involved: a single piece of evidence.

Cohen and Howe (1989) discussed the evaluation of large AI systems. They suggested several performance experiments that can be used for this purpose. One of the systems they discussed is

[5] In fact, this measure is related to hermeneutics and as such subjected to the debates about, and variations of, it (Mallery et al., 1986).

DOMINIC, a system for the routine design of mechanical devices (Howe et al., 1986). they show how performance evaluations guided their research through three generations of the system. The qualitative and the quantitative variants of this measure are discussed next.

Qualitative measure. This measure can be viewed as performing protocol analysis (Ericsson and Simon, 1980) on the system for evaluating its knowledge. The system is used to solve a variety of problems, and its problem-solving behavior is coded and analyzed. The analysis uses language different than that implementing the knowledge internally. Domain knowledge can further enhance the evaluation of the behavior and therefore, the knowledge evaluation.

This is one of the important techniques for evaluating systems (Cohen and Howe, 1989). It is often used in design research. The problem is that often, very few examples are provided and usually without substantial analysis.

Quantitative measure. This measure is based on the performance of a system over many problems that span the range of problems the knowledge of the system is expected to solve. The performance of the system is compared with the solutions generated by human experts or normative theories such as decision theory. The solutions can be also evaluated based on shared standards such as design codes. The latter is used in the example discussed later.

For example, Gaines (1989) used a functional quantitative measure to evaluate the initial state of knowledge of a learning program. The initial knowledge serves as a base-line for the learner and the amount of data necessary to generate a predetermined fixed performance level measures the quality of the initial knowledge.

In general, the measurement of performance of systems and their comparison is not straightforward. An example from performance evaluation of learning programs is discussed by Kononenko and Bratko (1991) . One of the causes for difficulty, they mention, is that the types of answers from different programs are not exactly the same.

Since this measure summarizes the results in statistics, it tends to hide some details of the system's behavior. Many controlled studies are needed to uncover behavior patterns that can be identified (although only qualitatively) by the qualitative functional measure.

4 Bridger

In order to ground the different evaluation measures and their tradeoffs it is necessary to demonstrate them in an evaluation of an existing system. The system selected is BRIDGER, a system developed to explore the potential of knowledge acquisition techniques for building design assistant for the preliminary design of cable-stayed bridges. First, we describe the bridge domain and then review the system's architecture and operation. Since the purpose of this paper is to focus on knowledge evaluation and not on BRIDGER, only the necessary parts required for the demonstration are described.[6,7]

[6] See (Reich, 1991a; Reich, 1991b; Reich, 1992) for further details.

[7] This demonstration can be percieved as a detailed case study (i.e., Adelman's (1991) third category of experiments discussed before). As such, it is a qualitative evaluation of the concept of knowledge measures discussed in this paper.

4.1 Domain of Cable-Stayed Bridges

Figure 1 describes the main components and dimensions of a cable-stayed bridge. A cable-stayed bridge is composed of a superstructure and a substructure. The superstructure is composed of a deck, towers, and stays that are attached to the towers and support the deck. The figure shows some of the properties which are used to describe a cable-stayed bridge; additional properties include: SPAN-N that is the number of spans of the bridge; DECK-A, DECK-MI, TOWER-A, TOWER-MI and STAY-A that are the cross-sectional areas and moment of inertia of the deck, tower, and the cross-sectional area of the stays respectively. In addition, DECK-M=1 denotes a concrete deck and DECK-M=2 denotes a steel deck. In a design scenario the requirements are expressed as a set of specification property-value pairs (e.g., the required length of the bridge). Design is then executed by making design choices as assignments of design-description property-value pairs (e.g., SPAN-M=500ft., STAY-N=20).

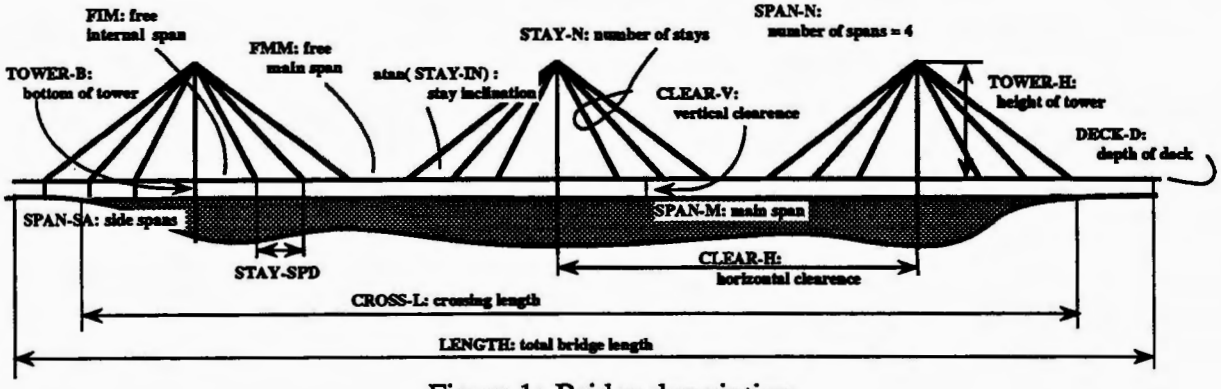


Figure 1: Bridge description

A rough illustration of the complexity of the domain can be conveyed by the number of properties used to describe various aspects of the problem: 9 properties describe a specification; 30 properties describe a design, 15 properties describe the analysis results, and 4 properties describe the evaluation of a bridge.

4.2 BRIDGER's Architecture

BRIDGER's architecture, shown in Figure 2, consists of two main systems: synthesis and redesign. The synthesis system is responsible for synthesizing several candidates from a given specification. Synthesis knowledge is generated by learning from existing designs and from successful design examples that are selected by the user. Since the knowledge created is heuristic by nature, candidate designs are usually inadequate in some aspects. The redesign system resolves this problem.

Candidate designs are transferred to a module that analyzes them and submits them to a redesign module, if necessary. The redesign module is responsible for modifying designs after their analysis. On receiving the analysis results, this module retrieves the best design modification for the bridge. The user can override the redesign modifications and supply explanations that enhance redesign knowledge. The results of the redesign system are acceptable designs. The designer evaluates the results and can submit a subset of them to the synthesis system for further training.

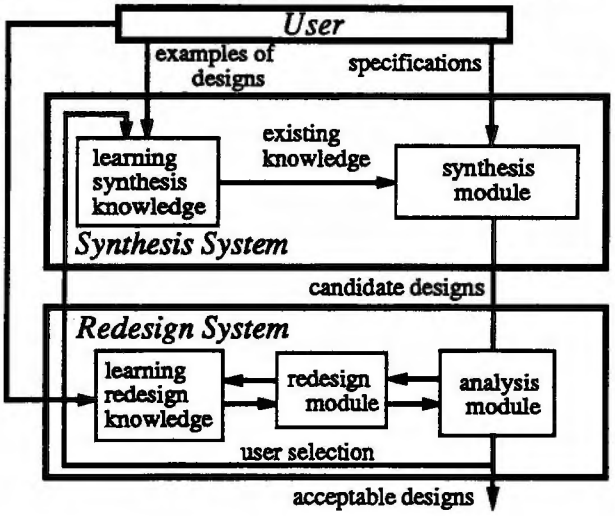


Figure 2: BRIDGER's architecture

ECOBWEB is the learning system that implements the synthesis system. It acquires synthesis knowledge and uses it to synthesize new bridges. ECOBWEB represents knowledge in a classification hierarchy. It has several operators that build the classification from examples. Learning and synthesis progress by using one-step look ahead search in the space of classification hierarchies directed by an evaluation function to select the best operator.

The evaluation function, called *category utility* (*CU*), evaluates a classification of a set of designs into mutually-exclusive classes $C_1, C_2, \ldots, C_n$, by:

$$CU = \frac{\sum_{k=1}^{n} P(C_k) \sum_i \sum_j P(A_i = V_{ij}|C_k)^2 - \sum_i \sum_j P(A_i = V_{ij})^2}{n} \quad (1)$$

where C_k is a class, $A_i = V_{ij}$ is a property-value pair, $P(x)$ is the probability of x, and n is the number of classes. The first term in the numerator measures the expected number of property-value pairs that can be guessed correctly by using the classification. The second term measures the same quantity *without* using the classes. Thus, the category utility measures the expected *increase* of property-value pairs that can be guessed *above* the guess based on frequency alone. The measurement is normalized with respect to the number of classes. The higher is the value of *CU*, the better the quality of the classification is.

BRIDGER has a variety of synthesis strategies, ranging from case-based to prototype-based design and from extensional to intentional strategies. To simplify the discussion we use the simplest strategy: an extensional case-based strategy. In this approach, BRIDGER *retrieves* a pre-determined number of candidate designs from the hierarchy. The candidates are complete descriptions of previously designed bridges. BRIDGER then *adapts* these candidates to fit the new specification by performing various scaling operations.

5 EXAMPLE

This section describes a detailed evaluation of BRIDGER's design knowledge as it develops through learning. Four hierarchies, K_1, K_2, K_3, and K_4, were generated. Hierarchy K_1 was generated from a set of original 96 bridge examples. Hierarchy K_2 was generated from the 96 examples after their analysis and redesign. Therefore it contains higher quality examples. Hierarchies K_3 and K_4 were generated from 144 and 192 good quality examples, respectively.

Our "unit" of knowledge will be a training example for ECOBWEB.[8] An additional example modifies knowledge and the subsequent performance. In the terminology of measurement theory, the relation on knowledge will be "better-than," and the binary operation is appending-hierarchies.[9] Since hierarchy K_1 is equal to K_2 in terms of the number of examples, we cannot treat them as distinct according to the selected unit of knowledge. On the other hand, K_1 is built of lower quality examples and cannot be compared meaningfully with K_3 or K_4. For reference purposes we maintain its evaluation.

5.1 Structural Measure

The structural content of knowledge was measured in two ways: quantitative and qualitative. The qualitative measure attempts to "understand" the knowledge generated from a domain perspective; and the quantitative measure is based on the evaluation function employed by ECOBWEB for assessing categories.

Qualitative measure. Figure 3 shows the K_2 synthesis hierarchy generated from 96 examples of bridges. The classes are described with some of their properties. Some properties are shown in bold font; these are the characteristic properties. Intuitively, characteristic property values of a class are those property values that are very common in the class and rarely appear in the other classes of the same level. The figure also shows the name of each class and in parenthesis the number of bridges used to generate it.

The hierarchy is subdivided into two large subclasses: class C which contains long bridges (i.e., long LENGTH and SPAN-M properties) with many stays, and class B which contains short bridges with fewer stays. Further subdivisions mainly reflect differences in the LENGTH, CROSS-L, SPAN-M, SPAN-N, and DECK-M properties. Several patterns emerge in the hierarchy. They can be interpreted using domain knowledge, and may point to some design heuristics. The number of examples, however, is not sufficient to allow learning to discover strong patterns; any explanation should cautiously be accepted.

For example, class B contains only bridges with steel decks and class C contains 44% concrete-deck bridges; in addition, the average main span of class B is shorter than that of class C. These trends point to a preference for using concrete for longer bridges and steel for shorter bridges. The first is correct, but the second is not.[10] A close look at the subclasses of C shows that H and I, which

[8] Here, we can see the theory-laden nature of evaluation: Considering an example as a unit of knowledge is natural to ECOBWEB, but may not make sense for other systems.

[9] Appending hierarchies is done by taking one hierarchy and training it by the examples used to generate the second hierarchy. This is an order dependent procedure that causes a violation of a commutative property of measures.

[10] See Table 3.3 in (Podolny and Scalzi, 1986) showing lower bids for concrete bridges as opposed to steel bridges

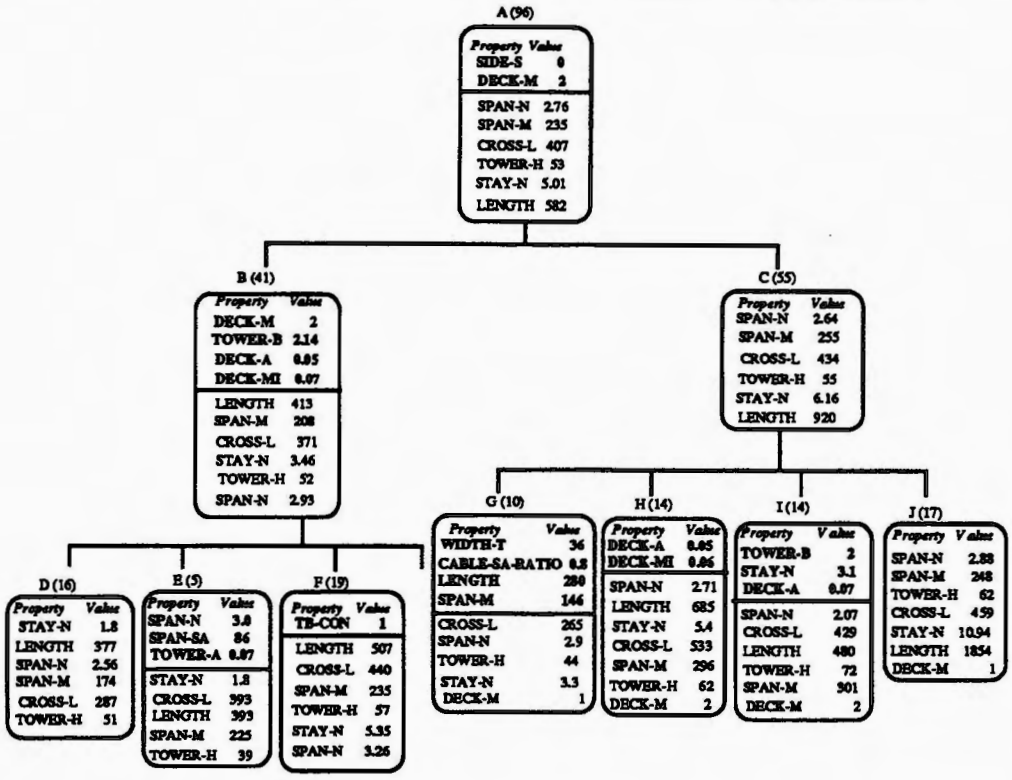


Figure 3: K_2 synthesis knowledge base

contain only steel bridges, have longer average main span than the two other classes, containing mainly concrete-deck bridges. Therefore, the preferences stated before no longer apply. The conclusion is that the average main span value of C is only a common value of the class but does not necessarily provide a good characterization of the class.

This example demonstrates the subjective and imprecise nature of the qualitative structural measure. It is not clear which aspects in the class description are important and how they should be interpreted. It is not clear how the *value* of the measure is to be defined. Is it by a carefully controlled statistical study in which different experts rate the knowledge? is it by counting "interesting" statements made about the hierarchy?[11]

Observing the content as the hierarchy grows can potentially explain some of the design behavior revealed by the other measures. Figure 4 illustrates the growth pattern of the synthesis hierarchy. It reflects the organization of knowledge rather than its content. Initially, the hierarchy is "flat," consisting of the root node and its leaves. When additional information is accumulated, a second level starts to grow. Approximately twice the number of examples is required to form that second level. This pattern of growth continues later.

The design performance (i.e., quality and time) is not expected to improve continuously, but rather

for several recent long bridges. Also see Figure 3.8 of that reference showing that concrete is preferred to steel for short spans.

[11] Hermeneutics will reject the idea of quantifying this "measure" arguing that the qualitative knowledge evaluation should maintain its present nature and be elaborated further as the text proceeds.

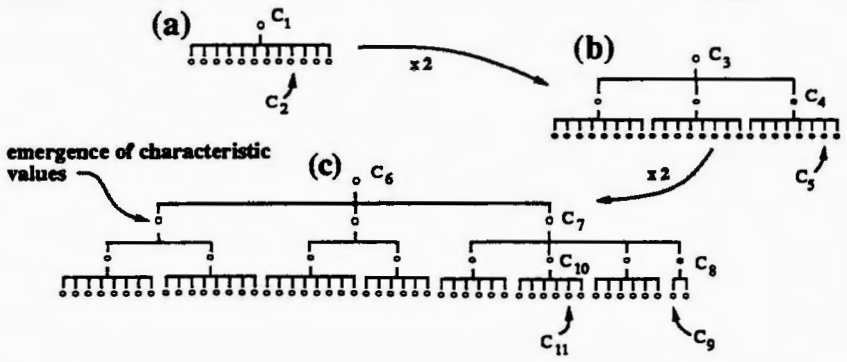


Figure 4: Qualitative description of hierarchy growth

in stages. To illustrate (see Figure 4), assume that a design is initiated with hierarchy (a) and that the best candidate is class C_2. If BRIDGER is asked to synthesize n candidates, it will consider all the sons of C_1 and output the n best matches to the new specification. After additional training, hierarchy (b) is generated and used for the same design; synthesis progresses from C_3 to C_4, and finally, to C_5. Now, synthesis chooses the best n sons of class C_4 as candidate designs. The sons of C_4 form a more homogeneous class than the sons of C_1, but this has required doubling the number of training examples. Additional training leads to the generation of of hierarchy (c). If synthesis progresses through the path C_6, C_7, C_{10}, and C_{11}, then candidates are generated from the sons of C_{10} (n out of the 6, assuming than $n \leq 6$). If the path ends at class C_9, the candidates are generated from the sons of C_7 (n out of the 20, assuming than $n \geq 2$). The first case will demonstrate an overall performance improvement, but the second will show similar performance quality and degraded time performance to that demonstrated by hierarchy (b). These differences suggest that learning is not continuous, although it may seem so when performance is averaged over many cases.

Summary of qualitative structural measure.
This measure is subjective and incomplete (i.e., no value is assigned to the measure). The evaluation can provide insight about the behavior of knowledge in design; mainly, since the internal mechanisms of the systems are known.

Quantitative measure. The quantity of knowledge can be defined by a measure, called *knowledge utility (KU)*, that calculates the *increase* in the number of properties that can be predicted for a given specification when using the *hierarchy*, relative to the number of properties that can be predicted by using values' frequency. The category utility function (*CU*) calculates the same item for a *classification*, not for a *hierarchy*. Applying *CU* recursively, starting from the root of the hierarchy, yields the desired utility measure:

$$\begin{aligned} &knowledge\text{-}utility\ (class):\\ &\quad if\ class\ is\ a\ leaf\ class,\ return\ 0.0;\\ &\quad else,\ return\ CU \times n + \sum_{k \in sons\ of\ class} P(C_k) \times knowledge\text{-}utility\ (son). \end{aligned} \tag{2}$$

After the calculation, the value is normalized by the number of properties describing artifacts.

When experience grows, the design knowledge emerging in the hierarchy is expected to converge to

better knowledge as shown in Figure 5(a). Starting from an arbitrary value influenced by the set of initial designs, the scope of knowledge increases as additional designs are learned. In principle, the larger scope allows for the generation of more designs; however, since new properties or property-value pairs have little association with previous property-value pairs, the quality of the generation will be poor. This initial growth of scope may reduce the quality of the existing knowledge. As more designs are learned, knowledge is accumulated on many possible combinations of property-value pairs and the utility increases.

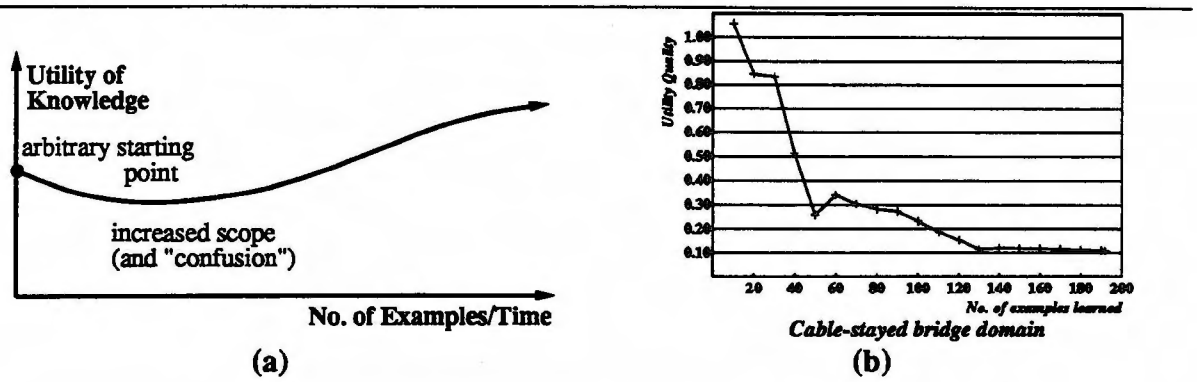


Figure 5: Improvement of design knowledge

Figure 5(b) shows the knowledge utility as a function of the examples learned by BRIDGER for the domain of cable-stayed bridges. The value of about 0.1 reached after learning 192 examples suggests that approximately 8 out of the 58 properties describing designs can be predicted accurately. This may seem to be a rather disappointing result since it is difficult to envision that a knowledge with such a low utility can be helpful in synthesis. It should be remembered, though, that knowledge utility calculates the utility *above frequency prediction*, and it averages this calculation over the complete classification, including its insignificant parts.

Summary of quantitative structural measure.
This measure is abstracted from the mechanisms that manipulate knowledge although it still relies on *CU* which certainly governs behavior. The bad measure observed here is in contrast to the good performance reported later and it demonstrates the difficulty in formulating good quantitative structural measures that can be used to predict performance.

5.2 Functional Measure

This measure evaluates knowledge in synthesis activities performed on 48 test specifications (see (Reich, 1991a) for details). The four knowledge hierarchies, K_1, K_2, K_3, and K_4, were used to synthesize 4 new candidates for each specification. The 192 synthesized bridges were used in the qualitative and quantitative functional evaluations.

Qualitative measure. Instead of analyzing the complete trace of synthesis, we focus the evaluation on one important synthesis step: the retrieval of candidate designs. Table 1 shows the number of different existing designs retrieved, and the names of the designs most commonly used. The

number of times each existing design was used in the generation of candidates, out of the 192 new candidates, is given below each name. The small number of designs retrieved reflects an internal tendency that characterizes the knowledge. In light of the discussion on Figure 4, it is not surprising that the structure of the hierarchy can lead to such behavior at the early stages of learning.

Table 1: Summary of retrieved designs

Knowledge	# of different designs	designs retrieved # of times retrieved (out of the 192)							
K_1	12	E10 45	E12 45	E49 45	E60 45				
K_2	8	E46 43	E55 43	E78 43	E91 43				
K_3	19	E2 16	E10 16	E32 16	E88 16	E19 12	E22 12		
		E24 12	E26 12	E29 12	E49 12	E115 12	E111 9		
K_4	19	E80 25	E144 25	E192 25	E3 15	E159 13	E162 13	E168 13	E135 13

The difference in the number of designs retrieved suggests a synthesis pattern similar to that presented in Figure 4. In particular, the synthesis pattern emerging from the K_1 and the K_2 hierarchies is probably similar to the path C_6, C_7, C_8, and C_9. Such a path forces the retrieval of designs from classes of designs higher in the hierarchy. Since a large class is used as a source of existing designs, the selection would usually favor a small number of 'strong' matches. If the path is from C_6 to C_{11}, the selection would be from smaller and different groups of classes, leading to the retrieval of a larger number of distinct designs. This is the case when K_3 and K_4 are used.

Figure 6 shows the four designs most often used when synthesizing with the K_2 hierarchy. These designs are listed in Table 1. The two designs on the right are scaled down by a factor of two. All four designs are two-span bridges with average main span (224 m). The range of spans is large, thus giving flexibility in retrieving a relevant design that do not require substantial scaling.

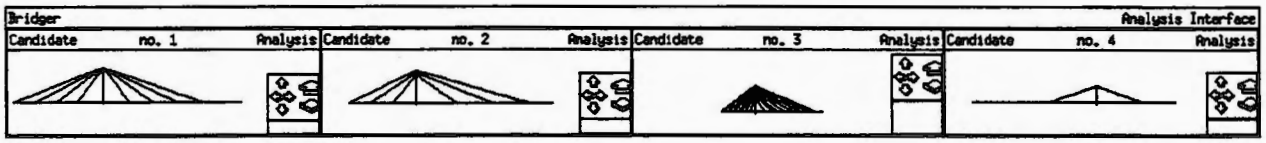


Figure 6: Functional assessment of K_2

Figure 7 shows the 12 designs most often used when synthesizing with the K_3 hierarchy. Most of them are three-span bridges. The average length of the main span is 179 m. A surprising observation is that most of the bridges have a small number of stays. This fact and the observation that almost no three-span bridges were used by the K_2 hierarchy point to the existence of a *shadowing* phenomenon. Certain bridges are not retrieved since they reside on hierarchy branches that are rarely visited. But once these branches become accessible, their leaves start being used as candidates.

Figure 8 shows the 8 designs most often used when synthesizing with the K_4 hierarchy. There is a better balance between two- and three-span bridges, and more variation in the number of stays.

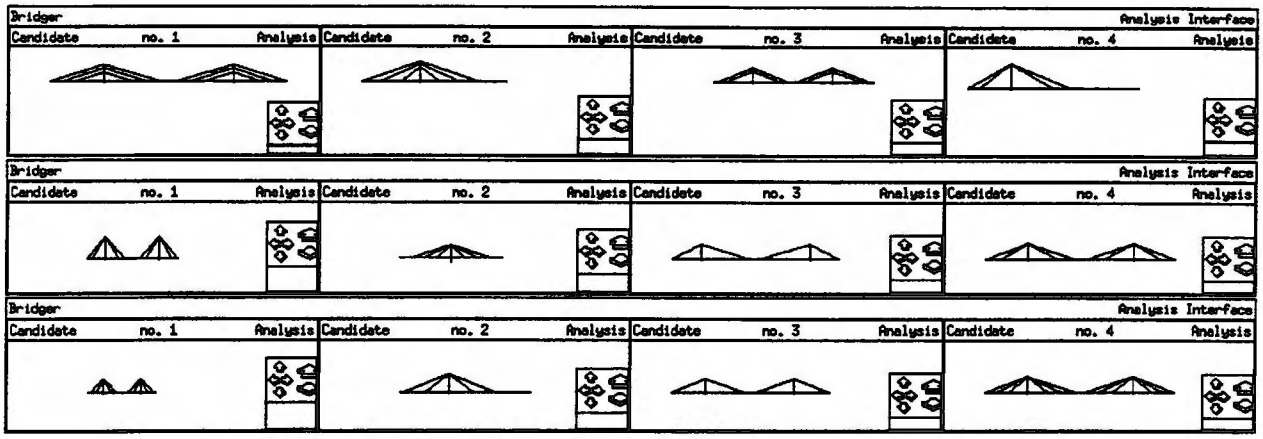


Figure 7: Functional assessment of K_3

The length of the main span of these designs is longer than before (345 m) and its variability is slightly less then that observed for the K_2 hierarchy. The increasing average length helps design large bridges without compromising the design of bridges with small spans.

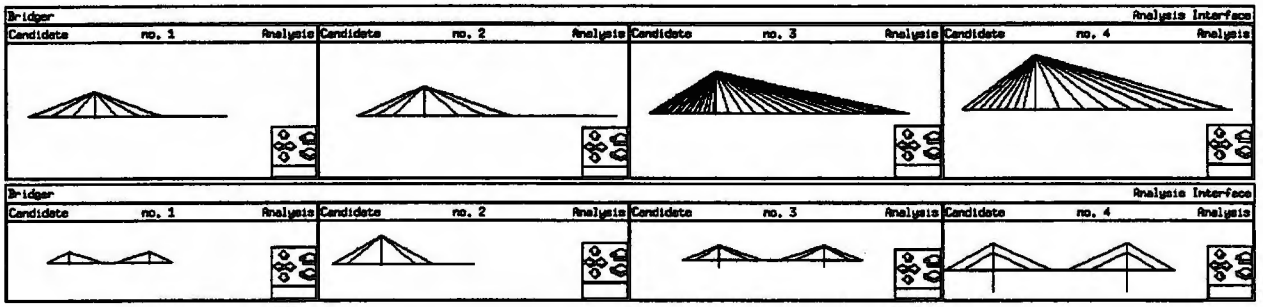


Figure 8: Functional assessment of K_4

Summary of qualitative functional measure.
This measure can be used to predict the quantitative measure by induction over behavior patterns. It can also be used to confirm the qualitative structural measure. A more detailed qualitative measure can point to important issues that need to be addressed in refining the system, for example, should the *shadowing* effect discussed before and confirmed here be handled? how to quantify the measure, by the number of designs retrieved, their quality, their variety?, etc.

Quantitative measure. This measure evaluates the design performance of BRIDGER while designing candidates for the 48 test specifications. Since the power of BRIDGER's synthesis process comes from two sources: (1) the retrieval of designs closely related to the new specification; and (2) the adaptation of candidates with scaling values; testing must address both.

The retrieval process is evaluated by the amount of *scaling* of the main span of the retrieved design needed to satisfy the new specification; it measures how close is the retrieved design to fulfilling the dimensional specification. The candidate adaptation process cannot be tested independently.

The combination of the two processes is tested by measuring the *quality* of candidate designs. The quality of a design is a weighted summation of the constraints that a design violates (Reich, 1991a).

The specifications used in the testing have considerable influence on the results. Specifications can reflect the average specification of the bridges designed thus far or be far from these average. Traditional expert systems will fail to perform on specifications that do not reflect their experience or range of operation, a phenomena called *brittleness*. The experiment also evaluates this aspect.

Table 2 provides the statistics of the *scaling* needed to adapt the candidates to the specifications of the 48 test problems and the *quality* of the designs synthesized. The columns denoted by *total* provide the average of these measures. The columns denoted by *lower*, *average*, and *upper*, provide the results for three groups of specifications corresponding to far lower than, similar to, or higher than, the average specification of existing bridges. These groups roughly divide the set into three equal parts.

Table 2: Scaling and Quality statistics of candidates

Knowledge	Scaling				Quality			
	lower	average	upper	total	lower	average	upper	total
K_1	1.25	3.58	5.85	3.074	1188.31	35.63	62.22	278.36
K_2	0.97	2.50	4.08	2.154	0.34	4.61	325.81	50.19
K_3	0.97	2.53	3.57	2.092	0.57	2.55	5.67	2.89
K_4	0.88	1.32	2.85	1.325	0.41	0.73	3.06	1.20

A MANOVA (Hays, 1988) analysis was performed to assess the statistical significance of the differences in the performance levels observed. The total scaling values satisfy: $K_2, K_3 >_{0.01} K_4$;[12] where the $>_{0.01}$ indicates that K_2 and K_3 are greater than K_4 with statistical significance at the $p < 0.01$ level and that the difference between K_2 and K_3 was not statistically significant. Therefore, *the more knowledge* BRIDGER *has, the more relevant are the retrieved candidates*. The improvement is not a smooth function, but occurs in steps as predicted before. The total quality values satisfy: $K_2 >_{0.01} K_3, K_4$; therefore *the more knowledge* BRIDGER *has, the better the quality of candidates it generates*. Similar results were observed for the lower, average, and upper ranges. In addition, the group of specification influences the results. The scaling values satisfy: *lower* $<_{0.01}$ *average* $<_{0.01}$ *upper*, whereas the quality values satisfy: *lower*, *average* $<_{0.01}$ *upper*. This confirms a known engineering heuristic stating that it is relatively easy to design artifacts that are similar to past experience or slightly scaled down and harder if designs are to be scaled up. Since designs are generated for every specification BRIDGER *does not exhibit a brittle behavior*.

The results above assume that performance is proportional to the amount of knowledge. However, the known power low of practice governing learning (Newell and Rosenbloom, 1981) suggests that performance varies as a power function of the number of examples. Another MANOVA analysis was run with this model. The results of the *scaling* remained as before, but, the results for the *quality* were more conclusive: $K_2 >_{0.01} K_3 >_{0.01} K_4$ and *lower* $<_{0.01}$ *average* $<_{0.01}$ *upper*. This further demonstrates the care in hypothesizing a performance measure and testing it.

[12] As mentioned before, the results of K_1 are not analyzed. If, however, we introduced K_1 into the analysis, the results of the scaling would be the same, but those of the quality would change to no statistically significant differences. This is due to the large variability in the quality of bridges designed with K_1. This example demonstrates the importance of executing a careful evaluation.

Summary of quantitative functional measure.
This measure is the most precise of all the measures, but it depends on the theory used to create it. Although, it tends to hide detailed information that can be useful in understanding the system behavior, additional detailed tests can be performed that reveal more performance characteristics.

6 DISCUSSION AND SUMMARY

Two definitions of knowledge have been offered, each leading to two types of quality measures. The four measures were demonstrated in the evaluation of the design system BRIDGER. The demonstration has several limitations. For example, the 48 test specifications where not generated randomly and some of the measures were partially executed. Nevertheless, the demonstration conveys several important aspects of knowledge evaluation. First, none of the measures alone provides a complete insight of knowledge quality; rather, the measures complement each other. Second, the type of measure defined or model posited (e.g., knowledge utility or power law of practice) has a significant impact on the success of the evaluation. Often, finding good models is a significant research problem. Third, the design of data collection experiments, whether qualitative or quantitative, can lead to interesting observations (e.g., the division of the specification set into three groups).

Table 3 summarizes the discussion. The *qualitative structural* measure is subjective and incomplete. Its use requires having substantial domain expertise. It can potentially predict the design behavior of a design system. It will be useful in systems with manageable size knowledge, otherwise it will be hard to execute. The *quantitative structural* measure is precise but detached from the mechanisms that manipulate knowledge. It may predict the system design performance if it is based on some of its mechanisms. Its execution requires the development of a formalism that will allow quantifying knowledge, this formalism will usually be system dependent. The *qualitative functional* measure can predict the quantitative measure and confirm the structural measure. It is important for revealing subtle issues in the behavior of a design system that need to be addressed in future development. The *quantitative functional* measure is precise, nevertheless, it loses details due to its tendency to summarize performance by statistics.

Table 3: Measures of design knowledge

	Structural		**Functional**	
	Qualitative	Quantitative	Qualitative	Quantitative
Type	subjective preliminary	precise incomplete	subjective imprecise	coarse complete
Require	domain expertise	precise formalism	expert decisions, test cases	computational evaluation
Used to	partialy predict performance	partially predict performance	predict performance	quantify performance
Apply on	simple representation, manageable size knowledge	everything subject to row 2	everything including black boxes	everything subject to row 2
Summarized by	textual information	quantitative data	textual + quantitative information	concise quantitative data

This paper does not advocate for the use of a single definition of knowledge or one type of

measure; it only states that the definition adopted and the measure used should be consistent, that is, the structural evaluation should be used for assessing the quality of knowledge if the structural definition of knowledge is adopted. In fact, beside this consistency, it is optional and useful to exercise all the four measures. Each of them has some value and feedback to provide and they all complement one another.

Table 4 summarizes the different measures according the terminology of measurement theory. A question mark denotes an unspecified entry. The two qualitative measures are incomplete since the mapping between knowledge to textual information is not quantifiable. The quantitative measures are complete but are not fully articulated. For example, the functional quantitative measure employs addition as the binary operation, but does not specify what exactly is measured by the real numbers. For example, to satisfy the addition operation in the example, the knowledge unit should reflect the logarithm of the number of examples and the measure should evaluate the logarithm of the performance. Therefore, we argue that not one of the measures is, or could be, "objective."

Table 4: Observed and formal systems

	Observed	Formal			
		Structural		Functional	
		Qualitative	Quantitative	Qualitative	Quantitative
Set	knowledge hierarchies	text	positive real numbers	text + numbers	positive real numbers
Relation	better-than	?	>	?	>
Operation	appending	?	+	?	+

Future work includes completing several important tasks. The first task is the collection of data on evaluations of additional design systems. These cases might be generalizable, thus, used to develop a methodology for the evaluation of design knowledge. Another task deals with the formation of guidelines for system development that will facilitate appropriate evaluation. Finally, research should address the evaluation of knowledge embedded in a design setting (i.e., cooperation between a human designer and its design assistant), thereby leading models reflecting actual design settings.

ACKNOWLEDGMENTS

This work has been supported in part by the Engineering Design Research Center, a National Science Foundation Engineering Research Center.

REFERENCES

Adelman, L. (1991). Experiments, quasi-experiments, and case studies: a review of empirical methods for evaluating decision support systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(2):293–301.

Balkany, A., Birmingham, W. B., and Tommelein, I. D. (1991). A knowledge-level analysis of several design tools. In Gero, J., editor, *Artificial Intelligence in Design'91, Proceedings of The First International Conference on Artificial Intelligence in Design, Edinburgh, UK*, pages 921–940, Oxford, UK. Butterworths.

Bijl, A. (1987). Strategies for CAD. In *Intelligent CAD Systems I*, pages 2–19. Springer-Verlag, Berlin.

Boulton, D. M. and Wallace, C. S. (1973). An information measure for hierarchical classification. *The Computer Journal*, 16(3):254–261.

Brown, D. C. and Spillance, M. B. (1991). An experimental evaluation of some design knowledge compilation mechanisms. In Gero, J., editor, *Artificial Intelligence in Design'91, Proceedings of The First International Conference on Artificial Intelligence in Design, Edinburgh, UK*, pages 323–336, Oxford, UK. Butterworths.

Clancey, W. J. (1988). Acquiring, representing, and evaluating a competence model of diagnostic strategy. In Chi, M. T. H., Glaser, R., and Farr, M. J., editors, *The Nature of Expertise*, pages 345–420. Lawrence Erlgaum Associates, Hillsdale, N.J.

Cohen, P. R. and Howe, A. E. (1989). Toward AI research methodology: Three case studies in evaluation. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-19(3):634–646.

Coyne, R. D., Rosenman, M. A., Radford, A. D., Balachandran, M., and Gero, J. S. (1990). *Knowledge-Based Design Systems*. Addison-Wesley, Reading, MA.

Doyle, J. (1988). Implicit knowledge and rational representation. Technical Report CMU-CS-88-134, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA.

Dreyfus, H. L. (1979). *What Computers Can't Do—The Limits of Artificial Intelligence*. Harper Colophon Books, New York.

Ericsson, K. A. and Simon, H. A. (1980). Verbal reports as data. *Psychological Review*, 87(3):215–251.

Fetzer, J. H. (1990). *Artificial Intelligence: Its Scope and Limits*. Kluwer Academic Publishers, Dordrecht, The Netherlands.

Gaines, B. R. (1989). The quantification of knowledge–formal foundation for acquisition methodologies. In Ras, Z. W., editor, *Methodologies for Intelligent Systems, 4*, pages 137–149, New York. North-Holland.

Gaines, B. R. and Shaw, M. L. G. (1989). Comparing the conceptual systems of experts. In *Proceedings of The Eleventh International Joint Conference on Artificial Intelligence*, pages 633–638, Detroit, MI. Morgan Kaufmann.

Gluck, M. and Corter, J. (1985). Information, uncertainty, and the utility of categories. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society, Irvine, CA*, pages 283–287, San Mateo, CA. Academic Press.

Green, C. J. and Keyes, M. M. (1987). Verification snd validation of expert systems. In *Proceedings of the Western Conference on Expert Systems (WESTEX-87)*, pages 38–43, Los Alamitos, CA. IEEE Computer Society Press.

Hays, W. L. (1988). *Statistics. Fourth edition.* Holt, Rinehart & Winston, New York.

Howe, A., Dixon, J., P., C., and M., S. (1986). Dominic: A domain-independent program for mechanical engineering design. *International Journal for Artificial Intelligence in Engineering*, 1(1):23–29.

Kononenko, I. and Bratko, I. (1991). Information-based evaluation criterion for classifier's performance. *Machine Learning*, 6(1):67–80.

Kreinovich, V. and Kumar, S. (1991). How to help intelligent systems with different uncertainty representations cooperate with each other. *Cybernetics and Systems*, 22(2):217–222.

Kyburg, H. E. J. (1988). Knowledge. In Lemmer, J. F. and Kanal, L. N., editors, *Uncertainty in Artificial Intelligence 2*, pages 263–272. North-Holland, Amsterdam.

Levesque, H. J. (1984). Foundations of a fucntional approach to knowledge representation. *Artificial Intelligence*, 23(2):155–212.

Mallery, J. C., Hurwitz, R., and Duffy, G. (1986). Hermeneutics: From textual explication to computer understanding. Technical Report A.I. Memo No. 871, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Cambridge, MA. Also apppeared in *The Encyclopedia of Artificial Intelligence*, John Wiley & Sonse, New York, 1987.

Michalski, R. S. (1986). Understanding the nature of learning: issues and research directions. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors, *Machine Learning: An Artificial Intelligence Approach, Vol 2*, pages 3–41. Tioga Press, Palo Alto, CA.

Mueller-Vollmer, K., editor (1985). *The Hermeneutics Reader*. Continuum, New York, N.Y.

Nazareth, D. L. (1989). Issues in the verification of knowledge in rule-based systems. *International Journal of Man-Machine Studies*, 30(3):255–271.

Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T., and Swartout, W. R. (1991). Enabling technology for knowledge sharing. *AI Magazing*, pages 37–56.

Newell, A. (1982). The knowledge level. *Artificial Intelligence*, 18(1):87–127.

Newell, A. and Rosenbloom, P. S. (1981). Mechanisms of skill acquisition and the power law of practice. In Anderson, J. R., editor, *Cognitive Skills and Their Acquisition*. Erlbaum Associates, Hillsdale, N.J.

Podolny, W. and Scalzi, J. B. (1986). *Construction and Design of Cable-Stayed Bridges. Second edition*. John Wiley and Sons, New York.

Reich, Y. (1991a). *Building and Improving Design Systems: A Machine Learning Approach*. PhD thesis, Department of Civil Engineering, Carnegie Mellon University, Pittsburgh, PA. (Available as Technical Report EDRC 02-16-91).

Reich, Y. (1991b). Design knowledge acquisition: Task analysis and a partial implementation. *Knowledge Acquisition*, 3(3):237–254.

Reich, Y. (1992). The acquisition and utilization of basic aesthetic criteria in design. *Artificial Intelligence in Engineering*, (accepted for publication).

Searle, J. R. (1980). Minds, brains, and programs. *The Behavioral and Brain Sciences*, 3:417–424.

Shortliffe, E. H. (1976). *Computer-based Medical Cunsultation: MYCIN*. Elsevier, New York, NY.

Skyrms, B. (1990). The value of knowledge. In Savage, C. W., editor, *Minnesota Studies in the Philosophy of Science: Volume XIV: Scientific Theories*, pages 245–266. University of Minnesota Press, Minneapolis, MN.

Stevens, J. (1946). On the theory of scales and measurement. *Science*, 103:677–680.

Zuse, H. and Bollmann, P. (1987). Software metrics: Using measurement theory to describe the properties and scales of static software complexity metrics. Technical Report RC 13504, IBM Research Division, Yorktown Heights, N.Y.

KNOWLEDGE ACQUISITION IN THE SMALL

Issues in Building Knowledge-Acquisition Tools from Pieces

Jay T. Runkel *William P. Birmingham**

Electrical Engineering and Computer Science Department

University of Michigan

Ann Arbor, MI 48109

USA

Abstract

The knowledge-systems community is interested in easing the system-development process. One approach is *mechanisms*, which view knowledge systems as a set of tasks, each of which can be realized by a computation mechanism. To be effective, knowledge-acquisition (KA) tools must be automatically configured. We present a method for doing this for model-based KA tools. The method advocates *KA mechanisms*, which acquire knowledge in the small, and a set of strategies that provide a global view of the KA activity. Without such a view, a KA tool will not efficiently interact with a domain expert.

1. INTRODUCTION

The knowledge-systems community has recently become interested in easing the knowledge-system development process. Several proposals for doing this have been offered, including translation languages (Neches *et al.*, 91) and what we shall term *mechanisms* (Marques *et al.*, 1992). The mechanism approach, with roots in generic tasks (Chandrasekaran, 1986), Steel's (1990) componential framework and problem-solving methods (PSM) (McDermott, 1988), views knowledge systems as a set of tasks, each of which can be realized by a computation mechanism. It has been postulated that many of these tasks are domain independent, or *reusable*. There has been some empirical evidence to support this postulation; a number of researchers have successfully ported systems from one domain to another by modifying only the domain-specific portions of the knowledge base (Birmingham and Tommelein, 1991; Langrana *et al.*, 1986; Johnson and Hayes-Roth, 1988; Maher, 1987; Brown and Chandrasekaran, 1987). Recently, an analytical study has demonstrated that common mechanisms exist among a number of configuration-design systems (Balkany *et al.*, 1991).

The advantages of mechanisms are the following. Once identified and made reusable, mechanisms can significantly reduce programming effort (Runkel *et al.*, 1992; Klinker *et al.*, 1990; Neches *et al.*, 1991). That is, mechanisms can be reused among a variety of systems. Furthermore, if mechanism behavior can be described in the language used by domain experts (practitioners), then they may be able to construct systems themselves, given appropriate tools (Klinker *et al.*, 1990).

To reap the potential advantages of mechanisms, software-development systems must be constructed to support their integration to form a complete knowledge system. Depending on the intended user– practitioner or knowledge engineer– the interface to such a system will need to be

* Contact author: Phone 313-936-1590; Fax 313-763-4617; email: wpb@zip.eecs.umich.edu.

customized. We are concerned with creating a software-development system, called DIDS[1], to help knowledge engineers construct systems faster by automating much of the requisite programming. Through an interactive process, a knowledge engineer using DIDS constructs a PSM from a library of mechanisms. This results in a knowledge system and a knowledge-acquisition (KA) tool. The KA tool is then used by a practitioner to develop a complete knowledge base.

Model-based KA tools are particularly well suited for this task. These tools have demonstrated the ability to create significant knowledge bases, in some cases without requiring knowledge engineer intervention. These tools exploit a model of the problem solver and its knowledge structures. The model describes how to acquire and structure knowledge. This model, however, is hand-crafted by a knowledge engineer familiar with the problem solver, is specific to a particular problem solver and knowledge structure, and highly compiled. This creates a fundamental problem: PSMs are built on a "*per application*" basis, so each DIDS-generated knowledge system will have a different PSM. Requiring a tool user to construct a detailed model of the PSM would abrogate any potential gains afforded by rapid programming.

Thus, a means for developing model-based KA tools without using hand-crafted, explicit models must be found. We propose that by combining knowledge of the problem-solving methods with properties of the underlying knowledge structure, this problem can be ameliorated. Specifically, we associate a set of *mechanisms for KA* (MeKA) (Marques *et al.*, 1992) with (PSM) mechanisms, and *strategies* and a "tool kit" with knowledge structures. A MeKA describes how to acquire a piece of knowledge independently of any PSM using its associated mechanism. An important element of a MeKA is a slot describing how to acquire knowledge, for example, deduce value A from B, or ask the user. Strategies define how those MeKAs should be assembled so that a meaningful interaction with a practitioner can take place, *i.e.*, questions are posed in an order that makes sense to the user.

Like mechanisms, MeKAs have the properties of reusability and combinability. This allows a library of MeKAs to built and reused among many knowledge-acquisition tools. Thus, the MeKA library reduces the time to create a knowledge-acquisition tool as a mechanism library does for a knowledge system.

In this paper, we expand on these issues, and present some initial ideals on developing model-based KA tools from mechanisms. Section 2 explains our framework of knowledge systems. KA mechanisms and the tool kit are described in Section 3. The use of strategies is presented in Section 4, with related work discussed in Section 5. A summary is given in Section 6.

2. The DIDS Framework

In this section we present the DIDS framework, which strongly affects our view of the KA task. We begin by defining the scope of systems we have studied so far.

2.1. Scope

The AI community has been active in design research for some time. This research has resulted primarily in design tools that model some aspect of a design task and achieve a reasonable level of performance. For example, systems such as R1 (McDermott, 1980), M1 (Birmingham *et al.*, 1989), and VT (Marcus *et al.*, 1987), have automated real design tasks, and have been beneficial through saving either time or money. Typically, these tools use knowledge-systems techniques for implementation.

---

[1] Domain-Independent Design System.

We are limiting our initial investigation of mechanisms to configuration-design systems. Roughly speaking, these systems[2] *construct* a design from a fixed set of parts, which may be stored as a library, or given as input. These parts are well characterized with respect to function and relationships among each other. During configuration, parts are selected, and then interconnected or arranged in a topological or sometimes geometrical order such that the user's specifications are met. Configuration design is an ideal starting point because the definition of the task is cogent, there are a large number of working systems that perform the task, and it is inherently diverse, *i.e.*, not all configuration tasks are the same.

2.2. Mechanisms and PSMs

Underlying our work is the notion that a configuration system can be represented as a collection of mechanisms. Many of these mechanisms can be separated from specific domain knowledge, and therefore made reusable. Mechanisms are made operational in a new domain by *adding* to them domain-specific knowledge.

Our framework of design, called *DIDS*, is composed of mechanisms and a standard knowledge structure. The framework is pictured in Figure 3-1. Mechanisms perform the actions necessary to perform the configuration task. The knowledge structure is a flexible knowledge-representation scheme specifically for design. The knowledge structure determines the correct mechanism grain *size* [Runkel *et al.* 1992].

Mechanisms must have two properties to be useful as system-building elements: reusability and combinability. Reusability ensures that the framework has some expressive power; that is, the framework does not degrade into a union of every design tool. A library containing reusable mechanisms will be small and manageable since relatively few mechanisms will be required to implement all configuration design tasks. Combinability allows mechanisms to be joined.

PSMs can be easily constructed from mechanisms with these properties. An example PSM and set of mechanisms is shown in Figure 3-2. The system works by extending the design (*e.g.*, selecting a part from a library), determining if the extension violates constraints, and then fixing any violations. The problem-solving steps are numbered, and the mechanisms implementing those steps are in boldface. While mechanisms represent fine-grained subtasks, PSMs correspond to higher-level tasks.

Mechanisms perform two basic subtasks: design modification and program control. Design modification, as its name implies, makes some change in the design. This can be adding a part, performing backtracking, and so forth. Program-control mechanisms sequence other mechanisms. For example, the loop implied by Step 8 in the PSM shown in Figure 3-2 is implemented by a control mechanism. Such control is necessary to sequence mechanisms to perform the overall design task.

The design tasks perform by mechanisms are at a level significantly higher than the tasks performed by constructs provided by design-programming languages, such as DSPL (Brown and Chandrasekaran, 1987), EDESYN (Maher, 1987), and DESCRIBE (Mittal and Araya, 1987). These languages can be used to implement a mechanism library. Because they encapsulate a particular function in the problem-solving process, mechanisms are likely to be more easily recognized by domain experts than *statements* in a design language. Because of their higher level of abstraction, only a few mechanisms will be needed to compose a problem-solving method, thus their use can substantially speed-up programming.

---

[2] See Mittal and Frayman (1989) for a formal definition.

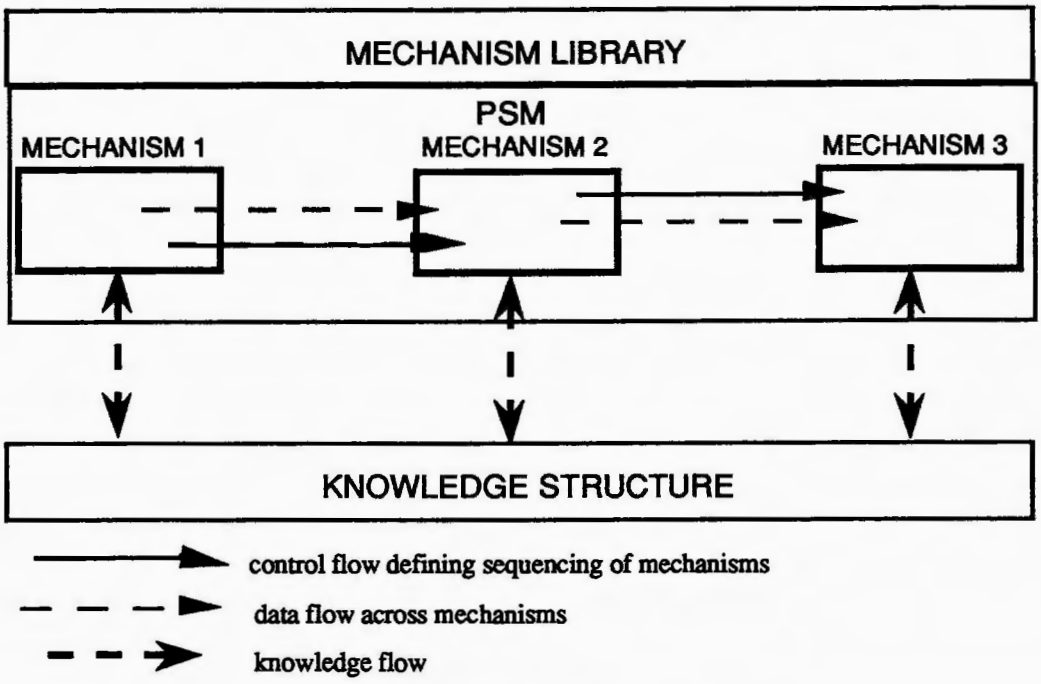


Figure 3-1: The DIDS design framework.

Mechanisms can be formulated in numerous ways, very few of which will result in reusable and combinable mechanisms. To guide mechanism formulation, we claim that:

mechanisms defined as operators on knowledge
+
tasks performed by mechanisms do not overlap
+
standard representation of knowledge types

$\Rightarrow$ reusable and combinable mechanisms

A mechanism is viewed as an *operator* activated during the design process when it is supplied domain knowledge of the requisite types. This view places no restrictions on the specific domain concepts that must be supplied, or on the source of domain concepts, thereby ensuring that a mechanism can be applied to any configuration task where the domain contains the appropriate knowledge types. Therefore, a mechanism defined in this manner is likely to be reusable.

To facilitate identification of reusable mechanisms, a knowledge-level analysis of six design systems was performed [Balkany *et al.*, 1991]. One of the results of this analysis was the identification of the types of knowledge that form the knowledge structure of design systems. We have identified nine types that recur in systems and that are sufficient to categorize all the knowledge used by the systems that we have studied. These types are listed in Appendix A. (It is expected that more types will be discovered by future analysis of other systems.) Thus, the DIDS framework incorporates a standard knowledge structure, realized as a set of types. A mechanism is defined by the operation it performs on a subset of the knowledge types.

The knowledge structure of a design system modeled in the DIDS framework contains a set of *nodes*, each consisting of one or more knowledge types. These nodes are linked to form a graph that represents a class of domain concepts. Instances of the nodes represent specific concepts, and

links between them express their interrelationships. For example, M1 contains *templates*, which consist of constraints that form preconditions, required-function knowledge, and a set of connections. An instance of a template describes the connections among several parts.

1. Extend the design and identify constraints on the extension just formed.
   (a) **extend-design**
   (b) **find-constraints**
2. Identify constraint violations; if none, go to step 8.
   check-for-constraint-violations
3. Suggest potential fixes for a constraint violation.
   suggest-fixes
4. Select the least costly fix not yet attempted. If there are no more fixes, quit with failure.
   rank-select-fixes
5. Tentatively modify the design and identify constraints on the modification just formed.
   (a) **make-modification**
   (b) **find-constraints**
6. Identify constraint violations due to the revision; if any, go to 3.
   check-for-constraint-violations
7. Remove relationships incompatible with the revision.
   propagate-changes
8. If the design is incomplete, go to step 1, else quit with success.
   test-if-done-variable-values-assigned

Figure 3-2: Example PSM and mechanisms (from Balkany *et al.* 1991).

A typical mechanism is given a node, containing several knowledge types, as an input. It inspects this node, and perhaps others linked to it, to make a design decision (*e.g.*, select a part from a library). Thus, mechanisms inspect knowledge and traverse the graph. The operation performed by the mechanism typically modifies the design. We restrict mechanisms to perform only one design modification.

Every configuration system, by definition, must have a library of parts that it assembles to build an artifact. This is the primary type of knowledge in the knowledge structure, and will be used to drive knowledge acquisition. The next three knowledge types –functional decomposition, task decomposition, and required function– define the conceptual objects reasoned with during the design process. The remaining knowledge types– constraints, preferences, ordering, connection, and arrangement– enable the problem solver to search the knowledge graph efficiently, to compute the values of design parameters, to connect and arrange parts, to detect when errors have been made, and to recover from those errors.

Figure 3-3 depicts a small portion of an example knowledge structure, representing the parts of a bicycle for a bicycle-configuration system. The solid lines are links representing functional relationship between nodes. Task, and required function links are not shown. Two of the nodes, bicycle and frame, have been expanded to show their attributes, and the constraint relationships between them. For simplicity, the other seven knowledge types have been not been included in this figure. Each of these knowledge types, however, is represented by a data structure that is included as part of a node, or by a structure that links the components of several nodes. The process used by DIDS to acquire the knowledge represented by graphs of this form is described in the next two sections.

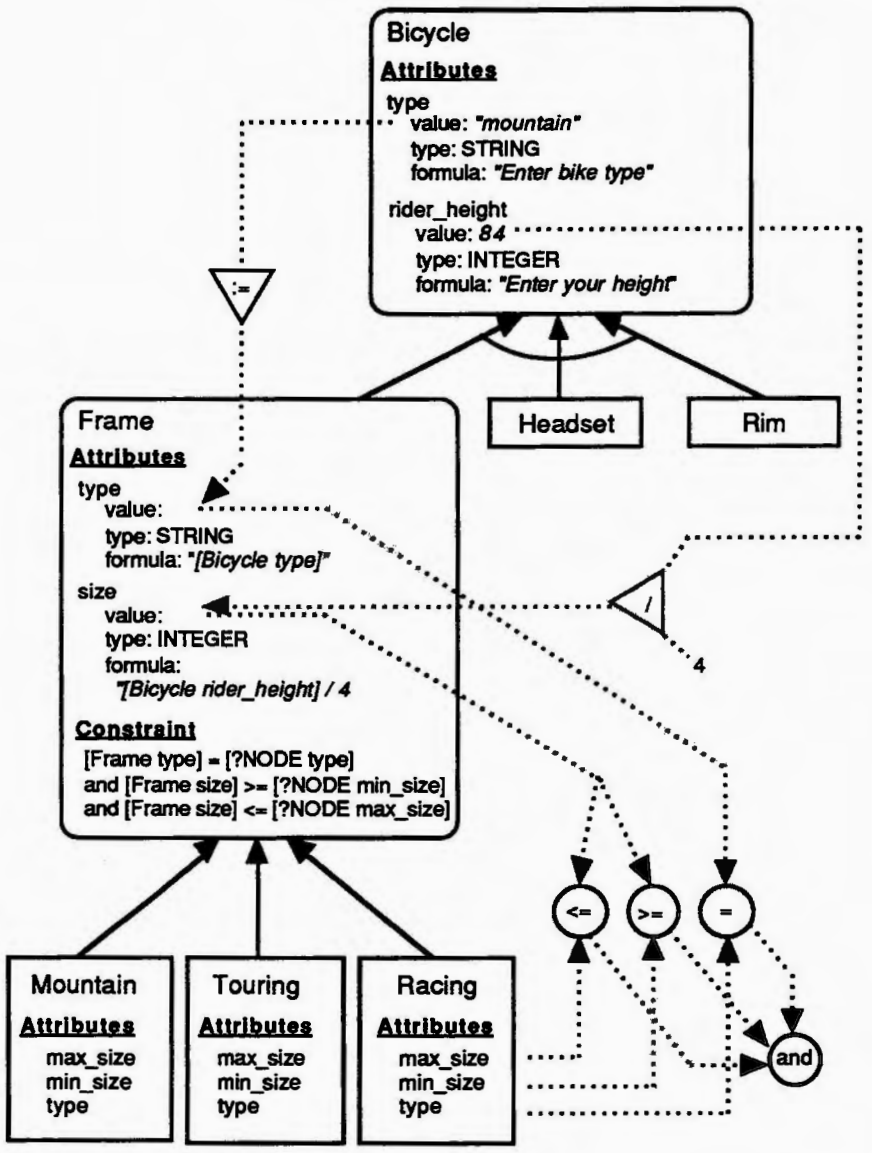


Figure 3-3: A portion of the knowledge structure for a bicycle-configuration system.

3. Mechanisms for Knowledge Acquisition

3.1. The Local Perspective

The DIDS-generated KA tool consists of two components: a knowledge-acquisition method (KAM), and a set of MeKAs. DIDS's mechanism library contains a MeKA for each mechanism, which describes how to acquire the domain knowledge used by that mechanism. The KAM plays the same role that the PSM plays in the problem solver; it determines how to sequence the MeKAs. The purpose of this sequence, however, is to produce meaningful dialog with the domain expert. A KA tool can be constructed by simply sequencing the MeKAs associated with the mechanisms in the PSM. This approach, however, does not result in a useful KA tool, because domain experts expect the KA tool to ask questions in a logical order. A tool that jumps among topics is difficult to

use and to understand. Thus, generating a KA tool requires a global view of the knowledge structures needed and their dependencies. MeKAs that acquire similar information must be arranged so that they can prompt the expert for all the information required at the same time. In addition, the situations when each MeKA should be invoked must be determined using a global model of the knowledge-acquisition process.

A MeKA has a local perspective of knowledge acquisition. For example, a mechanism may require that the node taken as an argument be linked to several other nodes of the same type, or that the node contains a specific knowledge type. The local perspective, however, prevents it from assuming anything about the structure of the graph that these nodes form. For example, a mechanism could not assume that the node graph was acyclic, a tree, or connected. Each problem-solving method, however, defines the structure of the graph it operates over. This structure results from the sequence of mechanisms that the PSM used.

3.2. MeKA Tool Kit

To support knowledge acquisition, every DIDS-generated KA tool contains a knowledge-acquisition tool kit, which has two responsibilities. First, it acquires the part library and the task or functional hierarchy. This information is required of all configuration design tools, but is assumed to exist by each MeKA. Thus, they are incapable of acquiring it. Second, the tool kit provides a set of specially designed interfaces for presenting and acquiring each knowledge type. These interfaces include:

1. *interactive dialog* - A text-based strategy where the user is asked a question, and responds by selecting from a menu or typing a response.

2. *graphical*- A graphical presentation strategy where the knowledge base is presented to the expert who modifies or extends it.

3. *table* - A cross between interactive and graphical strategies where a partially complete table is presented to the domain expert who fills in the missing elements.

3.3. The Form of MeKAs and KAMs

Every MeKA takes an argument, called the *focus*, which describes that portion of the knowledge structure it uses. Figure 3-7 shows several MeKAs, where each one acquires knowledge about the focus using a process that was defined by the MeKA's builder (a knowledge engineer). A MeKA begins by determining if it can use already acquired knowledge to infer new knowledge. If so, the MeKA performs the inference. Otherwise, the MeKA must acquire the knowledge from the domain expert. Acquiring knowledge is a three step process. First, the MeKA presents information that the user will likely want to review before entering the new knowledge. Next, new information is acquired from the user. Finally, the MeKA verifies that the knowledge provided by the expert satisfies a set of verification criteria.

A MeKA has four components -- infer, present, acquire, and verify -- which correspond to the four steps used to acquire knowledge for a mechanism. The infer component uses a MeKA specific inference procedure to automatically derive the necessary knowledge. The present component of a MeKA acts as a filter, presenting only the relevant elements of the knowledge base to the domain expert. The MeKA uses the tool kit to display a subset of the knowledge base, which is determined by the focus. The information displayed provides enough details to give the domain expert the appropriate context for the knowledge being requested, without overwhelming him with the complexities of the knowledge base. The tool kit displays the node in terms of the knowledge types being acquired by the MeKA.

Three strategies can be used to determine the display area:

- *present overview then refine* - Presents a high-level picture of the knowledge base, and then allows the domain expert to successively zoom on a smaller area.
- *current context* - Presents the area of the knowledge base that was presented by the previously executed MeKA.
- *incomplete area*- Jumps to the area of the knowledge base that is incompletely specified in terms of the knowledge acquired by the MeKA.

The acquire component either asks the user to modify the display of the present component, or asks the user about the portion of the knowledge base displayed by the present component.

The verify component determines if the knowledge supplied by the expert satisfies the restrictions placed on the knowledge by the mechanism. The verification checks performed by MeKAs fall into one of three classes. First, MeKAs ensures that the structure of the knowledge base matches the structure assumed by its associated mechanism. For example, the MeKA associated with a backtracking mechanism would determine whether backtracking is needed based upon the form of the constraint graph. Second, MeKAs ensure that the expert has supplied all the information required by its associated mechanism. For example, if the MeKA displays a table, the verification component would ensure that the domain expert has filled in the appropriate entries. Finally, MeKAs check the accuracy of the information supplied. For example, if the MeKA acquires a set of constraints used for part selection and the constraints refer only to tasks and not parts, then the MeKA will warn the expert.

3.4. An Example

The following example, where a bicycle-design system is to be generated (Runkel *et al.*, 1992), demonstrates some of the weaknesses of KAMs based solely on the sequencing of the mechanisms in the PSM. In this scenario, a user begins by constructing a PSM to perform bicycle design. The user selects an archived PSM that appears to solve his problem. DIDS then automatically generates the code for both the PSM and the KA tool. The designer uses the KA tool to construct a small portion of the knowledge base so that the PSM's performance can be evaluated. During knowledge acquisition, the MeKAs verification components may find situations where acquired knowledge does not match the assumptions made by the PSM. In addition, the designer may notice problems when testing the design tool. If these problems are found, they will require modification of the PSM, causing a new KA tool and PSM to be generated. This process continues until the PSM performs to the designer's expectations.

The PSM selected by the designer in this example, shown in Figure 3-4, maintains a queue of parts that are needed in the bicycle, but that have not been completely designed. The PSM assumes that the KA tool has organized the part library into a functional hierarchy consisting of OR and AND nodes. It begins each design cycle by computing the values of the attributes of all the parts in this queue. The *compute_spec_values* mechanism, which uses formulas associated with attributes, calculates the attributes' values. During each design cycle, the PSM designs the first node in the queue, called the *action*. If the action is an AND node, then the *select_all_parts* mechanism adds all of action's children to the queue. If the action is an OR node, then the *select_candidate_parts* mechanism determines the subset of action's children that satisfy the constraints. Then, the *select_best_part* mechanism selects the child that has the lowest cost. Finally, this part is added to the queue.

```
Queue readytasks;                    //queue of parts, already part of the
                                     //artifact, that need to be designed
DesignState ds;                      //datastructure containing the artifact
Node action,                         //the part being designed
     newpart;                        //the part selected as the design of action.
Set candidates;                      //set of parts that satisfy the constraints
Value node_type;                     //node type of action (OR, AND)

load_kb("bicycle.dids.kb");                  //loads the KB created by the KA tool
initialize_taskqueue(readytasks);            //Puts the first part "bicycle" into the queue
while (not_empty_queue(readytasks))
{
     // applies compute_spec_values mechanism to each part in the queue
     apply_to_queue(compute_spec_values, readytasks);
     get_next_task(readytasks, action);  //Gets the first part in the queue
     add_to_design(ds, action);          //Adds the part to the design
     if (is_an_AND_NODE(action))
     {
          select_all_parts(action, candidates); //selects all of action's children
          add_tasks(readytasks, candidates);    //and adds them to the queue
     }
     else if (is_an_OR_NODE(action))
     {
          //selects the subset of actions children that satisfy the constraints
          select_candidate_parts(action, candidates);
          select_best_part(candidates, newpart);          //selects the part with the least cost
          add_task(readytasks, newpart);       //adds the part to the queue
     };
}
```

Figure 3-4: The bicycle configuration PSM.

```
tool_kit: get parts
tool_kit: construct hierarchy
initial_taskqueue_MEKA
while (more parts in hierarchy to process)       // examine each node in the hierarchy
{
     apply_to_queue_MEKA;     //this and the next two MEKAs will cause redundant questioning
     get_next_task_MEKA;
     add_to_design_MEKA;
     compute_spec_values_MEKA;
     if Node is AND
          {
          select_all_parts_MEKA;
          add_tasks_MEKA;
          }
     else
          {
          select_candidate_parts_MEKA;
          select_best_part_MEKA;
          add_task_MEKA;
          }
}
```

Figure 3-5: The bicycle configuration KAM.

As just described, this PSM requires the following domain knowledge: a part library organized into a functional hierarchy, formulas associated with the attributes of each part, and a set of constraints that can be used to select parts at OR nodes. The KAM to acquire this knowledge is shown in Figure 3-5. The KA tool generated from the MeKAs associated with the mechanisms in the PSM

begins by invoking the tool kit's graph tool. The designer uses this tool to create the part library and to organize it into a functional hierarchy. This tool acquires *only* part names and any functional relationships among them. Figure 3-6 shows the functional hierarchy created by the graph tool.

Once the functional hierarchy has been built, the KAM orders the MeKAs in the same order as the mechanisms in the PSM. (Figure 3-7 shows pseudo-code descriptions for five of the mechanisms.) The first node designed by the PSM is the bicycle node, so it becomes the focus for the first sequence of MeKAs. The first mechanism executed in every design cycle is *compute_spec_values*. Therefore, the *compute_spec_values_MeKA* interviews the expert first. The MeKA displays the root node in the hierarchy, in this case the bicycle node, using a table format and prompts him for the specification attributes of a bicycle. The domain expert adds a new row to the table for each attribute of a bicycle. (Figure 3-8 shows the completed table.) The verify component of this MeKA requires that each attribute have either a value or a formula to compute its value. In this case, the formulas are questions put to the user to get the design parameters.

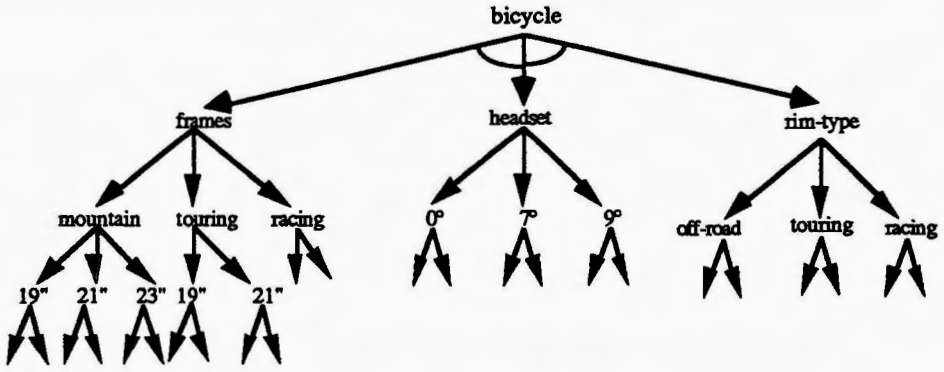


Figure 3-6: Functional hierarchy for bicycle scenario.

The next step during problem solving is to determine whether the focus is an OR or an AND node. The KAM makes this decision, and executes the *select_all_parts* MeKA since bicycle is an AND node. This MeKA does not acquire any knowledge, so the focus of the KA tool proceeds to the children of the bicycle node.

The next several steps of the KAM operate on the frame node. *Compute_spec_values_MeKA* asks the user to define the attributes of a frame. Figure 3-9 depicts the table created by the domain expert to describe the properties of a frame. At this point, however, the performance of the KA tool degrades due to the simplicity of the KAM. Since frame is an OR node the next MeKA is *select_candidate_parts_MeKA*. It acquires a constraint used to select the children of its focus. This constraint compares the attributes of the focus to the attributes of its children, and determines which children can be used in the design. Problems arise because the attributes of the children of frame have not yet been acquired. Therefore, the expert will have difficulty describing this constraint. Figure 3-10 shows the display presented to the user by the *select_candidate_parts_MeKA*.

4. Exploiting Properties of the Knowledge Structure

As shown in the previous section, simply integrating a set of MeKAs does not produce a tool that coherently interacts with an expert. Thus, we need to establish a set of acquisition strategies that provide the proper sequencing information. These strategies are derived from the knowledge types. In this section, we propose an initial set of strategies; this set will likely grow as we gain more experience.

Mechanism	MeKA

compute_spec_values(Node part)
{
For each attribute, att, of part
if att does not have a value then
compute att's value using its formula
}
	compute_spec_values-MeKA(*focus*)
	Knowledge used:
	Attributes
	Formulas
	Inference: none
	Prompt:
	Prompt("Use the following table to define the attributes of %s. For each attribute define either a value or a formula that can be used to compute the value of the attribute.", get_name(focus));
	Verify:
	{
	for all attributes, att, of focus
	ensure that att either has a value or att has a formula
	}

select_all_parts(Node nd, Set& candidates) {set candidates equal to nd's children in the functional hierarchy.}	select-best-part-MeKA(*focus*) { } */* no knowledge required */*

select_candidate_parts(Node nd, Set& candidates)
{
Get *nd's* children. in the functional hierarchy;
For each of child, *ch*, of nd
if ch satisfies the constraints on *nd* then
add *ch* to candidates;
}
	select-candidate-parts-MeKA(Node *focus*)
	Knowledge used: Constraints
	Present:
	present_attributes(focus);
	present_attributes(children(focus));
	Prompt:
	prompt("Enter the constraint used to select the from the possible designs of Frame %set", children(focus));
	Verify:
	{The constraint acquired constrains the focus and its children.}

select_best_part(Set& cand_parts, Node& best)
{
set best equal to the node in the set, cand_parts, that has the cost attribute with the smallest value.
}
	select-best-part-MeKA(Node *focus*)
	Knowledge aquired: Attribute: "Cost"
	Inferences: none
	Prompt:
	prompt("Enter the cost in dollars of the part %s?", get_name(focus));
	verify: cost is greater than 0;

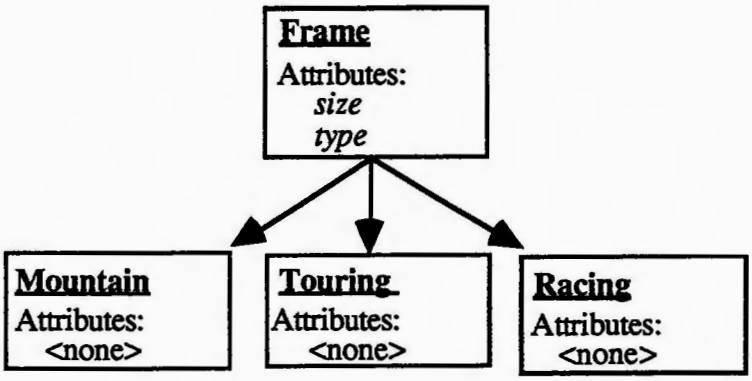
Figure 3-7: A few of the MeKAs in the bicycle design system.

Attribute	Value Type	Units	Value Range	Formula
type	*STRING*	*N/A*	*mountain, touring, racing*	*"Enter the type of bicycle you want to design:"*
person height	*INTEGER*	*inches*	*48...84*	*"Enter your height:"*
performance	*STRING*	*N/A*	*low, moderate, high*	*"Enter the type of bicycle performance you desire:"*

Figure 3-8: Table for acquiring attributes of bicycle.

Attribute	Value Type	Units	Value Range	Formula
size	*INTEGER*	*inches*	*19...27*	*[bicycle person height]/4*
type	*STRING*	*N/A*	*mountain, touring racing*	*[bicycle type]*

Figure 3-9: Table used to acquire the attributes of a frame.



Enter the constraint used to select the from the possible designs of Frame (Mountain, Touring, Racing):

Figure 3-10: Prompt for a constraint used by the select-candidate-parts MeKA.

4.1. Inherent Properties of the Knowledge structure

The knowledge structure assumed by the DIDS framework has inherent properties that can be exploited for knowledge acquisition. These properties are made operational in different ways, depending upon the mechanisms used. Thus, we will always need a MeKA for each mechanism to describe the specific way in which a piece of knowledge should be acquired and then represented in the knowledge system.

Recall that the DIDS framework views problem solving as sequencing mechanisms that operate on a knowledge structure. This knowledge structure is organized for each particular design system created by the DIDS tool, and not all elements of it may be present in all systems. There are, however, two types of knowledge– called the elemental types– that must be in all configuration systems: constraint and part library. This is because the fundamental configuration-design subtask is part selection. Thus, a configuration system must have a part library. The part-selection process requires constraints to compute values for part attributes and to bound the design space.

An initial set of heuristics for sequencing knowledge-acquisition tool behavior is given below.

1. Acquire (catalog) parts. These parts form the foundation of any system, and are easily acquired from catalogs. Parts also determine the types of attributes needed. There are a variety of methods to acquire parts, including user query using forms and optical scanning.

2. Acquire hierarchies and abstract parts. Depending on the type of problem-solving used, a functional hierarchy may be required. This hierarchy is formed out of *abstract* parts. These abstract parts must be specified by the user; This can only be done once the physical part library has been acquired. The abstraction process used to define abstract parts will, by definition, also define the hierarchy. The KA tool kit is designed specifically for this purpose. The tool kit supports both top-down and bottom-up hierarchy specification styles, as designers have different preferences.

3. Acquire support components and interconnection knowledge. These knowledge types usually go together, and thus should be acquired simultaneously. In addition, these knowledge types may help to define sections of a hierarchy, so they are often acquired simultaneously with abstract parts. Again, the KA tool kit is useful here, as this knowledge is normally graphically specified.

4. Acquire constraints and preferences. These are always defined relative to part attributes, so the parts must be given first. It is possible, however, to intersperse the acquisition of parts, abstract parts, and constraints and preferences. That is, some parts may be given, then some constraints, and then some more (abstract) parts, and so forth. Under some circumstances, interspersement may be preferable, as it could assist verification. This knowledge can be acquired through a dialog with the domain expert.

5. Acquire task knowledge. Most often, tasks are associated with parts (*e.g.*, design part X with specification (s_1, s_2, ..., s_n)). The type of knowledge usually acquired here is some specific partial ordering on tasks themselves (design X before Y), or a partial-ordering heuristic (*e.g.*, design big things before small things). In either case, the structure of the hierarchy should be known before task knowledge is acquired. Note: this knowledge can be acquired anytime after the hierarchy is established.

6. If a MeKA is used to acquire control, remove it from any loop. Many MeKAs will need to acquire knowledge solely for controlling the PSM. In general, this knowledge requires only a simple heuristic, and is not dependent on the particular state of the design, or a part in the library. Thus, it can be removed from any loops in the KAM. The first three mechanisms in Figure 3-5 have this property, and can be safely moved from the part-inspection loop.

We expect that the heuristics will be used in roughly the same order as given.

The Knowledge-Acquisition Tool Generator, a tool in DIDS (Runkel *et al.*, 1992), exploits these strategies. When a PSM is specified by a user, the MeKAs and the knowledge types are the known. This information is used by the tool generator to sequence the MeKAs to create a KA tool that can effectively interact with the domain expert. In essence, the tool generator creates a PSM for knowledge acquisition.

4.3. An Example

The KAM in Figure 3-5 is reorganized according to the strategies outlined in the previous section. The tool kit is used to acquire not only parts and hierarchy knowledge, but also attribute information. The tool kit exploits inheritance in the hierarchy allowing attributes to propagate up or down the hierarchy. This significantly reduces questioning.

The other MeKAs are also reorganized. Those performing program control are moved to the end of the acquisition session, and the information they require is asked only once. In addition, *add_tasks_MEKA*, which is visited for each node, has been moved to the end of the KAM.

Finally, the inner loop remains virtually the same, except as noted in the preceding paragraph. The constraint information for each part is still acquired in the loop. Since all the attributes have been acquired, however, the problems noted in Section 3 will not occur.

The result is a KA process that flows much more smoothly. The same type of knowledge is acquired at the same time, and in the correct order. Furthermore, redundant questions are eliminated.

```
tool_kit: get parts
    For Each Part:
        Acquire attribute information     // uses inherenitance to reduce questioning
tool_kit: construct hierarchy
while (more parts in hierarchy to process)       // examine each node in the hierarchy
{
    compute_spec_values_MEKA;
    if Node is AND
        select_all_parts_MEKA;
    else
        {
        select_candidate_parts_MEKA;
        select_best_part_MEKA;
        }
}
initial_taskqueue_MEKA
apply_to_queue_MEKA;
get_next_task_MEKA;
add_to_design_MEKA;
add_tasks_MEKA;
```

Figure 4-1: The new bicycle configuration KAM.

5. Related Work and Discussion

Klinker *et al.* (1990) have created a system called Spark, Burn, and Firefighter (SBF), which is based on combining mechanisms, similarly to DIDS. The two approaches differ in that SBF is geared towards non-programmers, the analysis of user's tasks is an integral part of system generation, and the task type is not restricted. This makes it difficult to determine *a priori* the types of knowledge and the set of mechanisms required to construct systems. Instead, they have developed a *shared* vocabulary of task activities that can be used to describe tasks in domain-independent terms. Burn, the KA component of SBF, constructs tools by combining computational KA mechanisms, which are analogous to DIDS's MeKAs. These mechanisms are ordered using the sequence of mechanisms in the PSM, similar to the method discussed in Section 3.

PROTEGE II (Puerta *et al.*, 1992) generalizes the capabilities of PROTEGE (Musen, 1989) and combines them with a mechanism-based model. PROTEGE II contains a library of both tasks and mechanisms. Links, which connect tasks to mechanisms, suggest mechanisms capable of performing a user's task. Protege II associates with each mechanism an arbitrarily complex data model, instead of a standard set of knowledge structures like DIDS. Each mechanism has a set of editors, similar to MeKAs, capable of acquiring its model. Once the mechanisms have been selected, PROTEGE II generates a KA tool by combining the editors associated with the mechanisms.

KADS establishes a methodology for constructing knowledge-based systems based upon a model of expertise similar to that of DIDS, but at a higher abstraction level. The KADS model of expertise contains domain knowledge, inference and task knowledge, and strategic knowledge, which are concepts that are analogous to knowledge structures, mechanisms, and PSMs, respectively. Like DIDS, knowledge systems are constructed by reusing computational constructs, and then encoding the domain knowledge in a form usable by these constructs. The level of automation and its scope distinguish KADS from DIDS. KADS defines a methodology, supported by tools, for constructing knowledge systems in any domain. DIDS, on the other hand, provides a greater level of automation to the process of building knowledge systems for a smaller range of tasks.

The process of building a knowledge system in Laps (Piazza and Helsabeck, 1990), differs significantly from that of DIDS. LAPS, which is a system geared towards diagnosis tasks, automates KA by beginning with a set of cases. The domain expert uses a table interface to acquire the possible symptoms, hypothesises, and intermediate decision steps. Laps, then tries to infer, with the knowledge engineer's help, the underlying knowledge structures and PSM by prompting the domain expert to decompose and to better specify the intermediate steps. Once the model has been determined, Laps drives the KA process to complete the knowledge base. DIDS, however, does not begin KA from a set of cases, but first asks the domain expert and knowledge engineer to define the PSM. DIDS then automatically generates a problem solver, infers the knowledge structures used by the problem solver, and a generates a KA tool using the PSM. As KA continues, the PSM is revised until the PSM and knowledge structures being acquired closely match the problem domain.

DIDS automates the process of building model-based KA tools for design tasks, such as CGEN (Birmingham and Siewiorek, 1991) and SALT (Marcus, 1988). Like these tools, DIDS uses a model of both the PSM and knowledge assumed by the PSM to derive its power. The nine knowledge types define the possible knowledge roles that must be acquired by the KA tool. MeKAs associated with each mechanism determine which of these knowledge roles are required, how to best acquire them, and how to generate code. The consistency and validation checks performed by these model-based tools are also present in DIDS-generated KA tools. The verification component of the MeKAs implements these checks.

DIDS-generated tools, at present, are restricted to the acquisition of declarative knowledge. Procedural knowledge is fully encoded in mechanisms, which are hand entered by a knowledge engineer; the KA tools do not support this process. Our experience has been, however, that all knowledge required to construct a configuration-design system is declarative. Thus, DID-generated design tools are not *knowledge starved.*

An issue with library-based systems, like DIDS, is organizing and describing a library to allow both easy addition and access to mechanisms. A particularly important issue is the unambiguous description of a mechanism's function. Certainly, the descriptions given in Figure 3-7 are not adequate for all mechanisms. The work described in this paper does not address this issue, although we are working on solutions.

DIDS shares many features with systems that automate the development of general software. DIDS falls between domain-specific program generators that automatically generate applications in a narrow domain, and systems that use massive code or program schema libraries, which cover broad domains, to assist programmers (Krueger, 1989). DIDS, like application generators, automatically generates source code. DIDS, however, does not have the severe narrow-domain restrictions that limit the usefulness of most application generators (Horowitz *et al.*, 1985). DIDS achieves this additional functionality by providing a library of basic elements (knowledge structures, mechanisms, and MeKAs) that represent the building blocks of a class of systems. The flexibility of the elements enables a broad range of systems to be generated, and the well-defined uses of these elements allows the generation process to be completely automated. Systems that reuse code or program schemas resort to a semi-automated development process due to the large number of elements in the library and small number of constraints restricting the selection and combination of these elements (Rich and Waters, 1990).

6. Summary

In this paper, we have presented an evolving methodology for building model-based knowledge-acquisition tools by pieces. These tools are constructed by assembling reusable software components, which we call MeKAs, that perform well-defined subtasks. The difficulty in building

tools in this way is finding an ordering of MeKAs that will produce a meaningful dialog with the user. We believe that this problem can be overcome by exploiting properties implicit in the knowledge structure assumed by the MeKAs, and by using a set of high-level strategies that are specific to design, the domain we are exploring, but independent of a particular design task.

We are currently implementing the DIDS system. A library of about 30 mechanisms has been developed, and is growing, and several design tools have been constructed. We are just beginning to develop the knowledge-acquisition tools, and have created a library of about 10 MeKAs. We have not yet built a system to automatically configure them.

Acknowledgments

Iris Tommelein, Alan Balkany, and the other members of the University of Michigan's Integrated Systems Design Group have made significant contributions to the DIDS research. This work was funded, in part, by the National Science Foundation Grant MIPS-905781 and by Digital Equipment Corporation.

References

Balkany, A., Birmingham, W.P., and Tommelein, I.D. (1991). A Knowledge-level Analysis of Several Design Tools, Proceedings of AI and Design '91, Edinburgh, U.K., Butterworth Scientific Ltd.

Birmingham, W. P., Gupta, A., and Siewiorek, D. (October 1989). The MICON System for Computer Design, *IEEE Micro.*

Birmingham, W.P. and Tommelein, I.D. (1991). Towards a Domain-Independent Synthesis System, Green, M. (ed.): Knowledge Aided Design, Academic Press, London.

Brown, D. and Chandrasekaran, B. (1987). Design Problem Solving - Knowledge Structures and Control Strategies, Morgan Kaufman, San Mateo, CA.

Chandrasekaran, B. (Fall 1986). Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design, *AI Magazine.*

Chandrasekaran, B. (Winter 1990). Design Problem Solving: A Task Analysis, AI Magazine.

Eshelman, L., Ehret, D., McDermott, J., Tan, M. (1987). MOLE: A tenacious knowledge acquisition tool, *International Journal of Man-Machine Studies.*

Frayman, F., Mittal, S. (August 1987). Cossack: A Constraint-Based Expert System For Configuration Tasks, *Proceedings of the 2nd International Conference on Application of AI to Engineering.*

Gero, J. (Winter 1990). Design Prototypes: A Knowledge Representation Schema for Design, *AI Magazine.*

Horowitz, E., Kemper A., Narasimhan, B. (January 1985). A Survey of Application Generators, IEEE Software.

Johnson, M.V.Jr and Hayes-Roth, B. (1988). Learning to Solve Problems by Analogy, Stanford University, Department of Computer Science, Knowledge Systems Laboratory, Report No. KSL-88-01.

Kahn, G., (1988), From Observing Knowledge Engineers to Automating Knowledge Acquisition, *Automating Knowledge For Expert Systems,* S. Marcus (ed.), Kluwer Academic Publishers.

Klinker, G. (August 1988). Knack: Sample-driven knowledge acquisition for reporting systems, *Automatiing Knowledge Acquisition for Expert Systems,* S. Marcus (ed.), Kluwer Academic Publishers.

Klinker, G., Bhola, C., Dallemagne, G., Marques, D., McDermott, J. (1990). Usable and Reusable Programming Constructs, *Proceedings of the 5th Knowledge Acquisition Workshop,* AAAI.

Krueger. C. (1989). Models of Reuse in Software Engineering, Carnegie Mellon University, Computer Science Tecnical Report, CMU-CS-89-188.

Langrana, N., Mitchell, T., Ramachandran, N. (1986). Progress Towards a Knowledge-based Aid for Mechanical Design, *Symposium on Integrated and Intelligent Manufacturing*, The American Society of Manufacturing Engineers.

Maher, M. (March, 1987). Engineering Design Synthesis: A Domain Independent Representation, *AI EDAM*.

Marcus, S. (1988). Salt: A Knowledge Acquisition tool for propose-and-revise systems, *Automating Knowledge For Expert Systems*, S. Marcus (ed.), Kluwer Academic Publishers.

Marcus, S., Stout, J., McDermott, J. (Winter 1987). VT: An Expert Elevator Designer that Uses Knowledge-based Backtracking, *AI Magazine*.

Marques, D., Dallemange, G., Klinker, G., McDermott, J., Tung, D. (June 1992). Easy Programming: Empowering People to Build Their Own Applications, *IEEE Expert*.

McDermott, J. (1988). Preliminary Steps Toward a Taxonomy of Problem-Solving Methods, in Automating Knowledge Acquisition for Expert Systems, Marcus S. (ed.), Kluwer Academic Publishers, Boston, MA.

McDermott, J. (1980). R1: A Rule-Based Configurer of Computer Systems, Department of Computer Science, No. CMU-CS-80-119, Carnegie Mellon University.

Mittal, S., and Araya, A. (August 1987). A Knowledge-Base Framework for Design, *Proceedings of the 9th IJCAI*.

Mittal, S., and Frayman, F. (August 1989). Towards a Generic Model of Configuration Tasks, *Proceedings of the 11th IJCAI*.

Musen, M. (1989). Automated Generation of Model-Based Knowledge-Acquisition Tools, Pitman, London.

Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R. , Senator, T., Swartout, W. (Fall 1991). Enabling Technology for Knowledge Sharing, *AI Magazine*.

Neighbors, J. (September 1984). The Draco Aproach to Constructing Software From Reusable Components, IEEE Transactions on Software Engineering.

Newell, A. (Summber 1981). The Knowledge Level, *AI Magazine*.

di Piazza, J., Helsabeck, F. (Fall 1990). Laps: Cases to Models to Complete Expert Systems, *Ai Magazine*.

Puerta, A., Tu, S., Musen, M. (1992). Modeling Tasks with Mechanisms, Knowledge Systems Laboratory, Stanford University, Report KSL-92-30.

Eriksson, H., Shahar, Y., Tu, S., Puerta, A., Musen, M., (1992). Task Modeling with Reusable Problem-Solving Methods, Knowledge Systems Laboratory, Stanford University, Report KSL-92-43.

Rich, C., Waters, R. (November 1988). Programmers Apprentice: A Research Overview, *IEEE Computer*.

Rich, C., Waters, R. (1990). The Programmer's Apprentice, Addison-Wesley, New York.

Runkel, J., Birmingham, W., Darr, T., Maxim, B., Tommelein, I. (1992). Domain-Independent Design System: Environment for Rapid Development of Configuration Design Systems, Artificial Intelligence in Design '92, Kluwer Academic Publishers, Gero, J. (ed.)

Sriram, D., Cheong, K. (1991) Engineering Design Cycle: A Case Study and Implications for CAE, Green, M. (ed.): Knowledge Aided Design, Academic Press, London.

Steels, L. (Summer 1990), Components of Expertise,*AI Magazine*.

Tommelein, I.D., Levitt, R.E., Hayes-Roth, B., Confrey, T. (January 1991). SightPlan Experiments: Alternate Strategies for Site Layout Design, ASCE, *Journal of Computing in Civil Engineering*.

Tong, C. (1987) Towards an Engineering Science of Knowledge-Based Design, *Artificial Intelligence in Engineering*.

Appendix A - The knowledge Types

The knowledge types are the following:

1. **Library of Parts** - The catalog of parts from which an artifact is built. The definition of a part includes a set of attributes and a set of ports. The ports of a part define where it can be connected to other parts.

2. **Functional Decomposition** - Functional-decomposition knowledge successively decomposes the artifact to be designed along functional lines. The root node of the hierarchy contains the most abstract, or general function, performed by the artifact, and the leaf nodes represent the physically realizable functions, *i.e.*, functions performed by parts. The hierarchy describes how abstract functions may be realized by combining sets of lower-level functions.

3. **Task Decomposition** - Design systems often contain task-decomposition knowledge that successively decomposes the task of designing an artifact into subtasks. This knowledge decomposes some task T into subtasks T1, T2, ..., Tn, which are smaller and easier to solve.

4. **Required Functions** - Parts and abstract parts often require functions performed by other parts to support their operation. Associated with each function performed by a part is a list of required functions that must be realized by the artifact. The part will not realize its intended functions unless the artifact realizes the required functions. Required functions do not add any desired functionality to the artifact; rather, they perform a function that is necessary for other parts to operate.

5. **Constraints** - Constraints specify algebraic relationships among the attributes of design concepts that must be maintained. Constraints enable the problem solver to distinguish acceptable from unacceptable solutions and to compute the values of the attributes of parts, tasks, and functions.

6. **Preference Knowledge** - Preference knowledge enables a design system to choose between sets of acceptable design alternatives. Preferences differ from constraints in that constraints eliminate alternatives, while preferences rank a set of acceptable alternatives so that optimal designs can be produced.

7. **Ordering Knowledge** - Problems solvers use task-ordering knowledge to determine the most efficient order of performing the tasks given by the task-decomposition knowledge.

8. **Connection Knowledge** - Connection knowledge specifies how parts can be connected. Every part contains a list of ports through which it can be connected to other parts. Connection knowledge constrains the set of possible connections that can be made among ports.

9. **Arrangement Knowledge** - Arrangement knowledge specifies how parts can be geometrically arranged. It constrains the topological relationships between parts in a design.

CASES, MODELS AND INTEGRATED KNOWLEDGE ACQUSITION TO FORMALIZE OPERATORS IN MANUFACTURING [1]

Gabriele Schmidt and **Monika Zickwolff**
German Research Center for Artificial Intelligence
Erwin-Schrödinger-Str.
W-6750 Kaiserslautern
Germany
e-mail: schmidt@dfki.uni-kl.de

Technical University Darmstadt
FB Mathematik
Schloßgartenstraße 7
W-6100 Darmstadt
Germany
e-mail: xmatdc0m@ddathd21.bitnet

Abstract. In the real world application domain of mechanical engineering a model-based knowledge acquisition approach was adopted. Task-specific models provide a useful guidance to begin the design of an acquisition framework, but they may not be sufficient for its elaboration. Therefore, important domain knowledge (cases) is additionally taken into consideration in the development of tools and their integration. The model-based and case-oriented tool COKAM+ by which operator definitions are acquired is used to illustrate this concept and its implications in detail.

1 Introduction

In the field of knowledge acquisition, there are several approaches to support the knowledge engineer in developing models of the problem solving process, e.g. the KADS methodology [Wielinga et al., 1991], generic tasks [Chandrasekaran, 1988] or role-limiting-method [McDermott, 1988]. In order to select a general model from a library or develop it, the knowledge engineer has to analyze the application domain carefully and do some abstractions. Hence, global knowledge about the domain and detailed domain knowledge is available which the model does not reflect. These data should also be utilized in the acquisition process by referring the domain knowledge to the model.
In our approach the model of expertise [Schmalhofer et al., 1991b] is utilized. It is a description of the human problem solving process and moreover a specification of the problem solving process of the future knowledge-based system. To meet both requirements our model has to be very general. Some important aspects of the domain are thus represented in the model only in an indirect manner.
Both, the model and the additional knowledge about the domain should influence the knowledge acquisition process. They can serve as a guidance for the development of the design of individual tools and an integrated knowledge acquisition framework.

[1]This research was funded as part of the ARC-TEC project by grant ITW 8902 C4 from BMFT (German ministry for research and technology).

The interactive knowledge acquisition tool COKAM+ (Case-Oriented Knowledge Acquisition Method from text) was designed based on these aspects. With this tool knowledge, i.e. operator definitions, can be acquired from text in a case-oriented and model-based manner [Schmidt and Schmalhofer, 1990, Schmalhofer and Schmidt, 1991]. The tool is embedded in our integrated knowledge acquisition framework[2], so that in interaction with the other tools in the integrated framework it can be employed to perform a very efficient knowledge elicitation and step-wise formalization process.

In the next section, the analysis of our domain reveals the problem solving process of the expert and the importance of the information source *cases* within this process. It is shown how the cases are described in the domain. The model of expertise derived from these results is briefly reported. In the following section, the interactions of the tools in the integrated knowledge acquisition method are summarized with respect to COKAM+. Then, the input information which COKAM+ obtains from the tool CECoS of the integrated framework is described in detail and improvements are suggested in order to obtain more formal information. In particular, a new method for analyzing conceptual data, the so called "Formal Concept Analysis" [Wille, 1989] can support representation and further formalization of the knowledge demanded by COKAM+. Section 5 shows the application of all guidances for an efficient formalization of operators. After showing application examples for the knowledge acquired with COKAM+ in section 6 the article concludes with a discussion and futures works.

2 Cases and a Model of Expertise in Manufacturing

Within mechanical engineering the concrete application task of our approach is the production planning for the manufacturing of rotational parts. For manufacturing a rotational part with a CNC–machine, each production step has to be planned. This is a demanding problem in mechanical engineering [Spur, 1979, Thien and Chien, 1985] which must be solved by experts.
Interviews and think aloud studies with the expert [Schmidt et al., 1990] showed that prototypical cases are very often consulted during the problem solving process of production planning. This result is confirmed by the literature of mechanical engineering: in 80 % of planning tasks the expert performs modification planning [Eversheim, 1989], i.e. prior solutions of cases are modified to obtain an appropriate new production plan. Thus, *cases* are a very important information source in our knowledge acquisition approach.
Cases of production planning are used in COKAM+, in which especially the operators of these cases are examined. In the following, the real world operator descriptions in mechanical engineering are shown. Since COKAM+ is also model-based, the model of the problem solving process is briefly presented.

2.1 Real World Case Descriptions in Mechanical Engineering

Cases are available in filing cabinets of companies or sometimes in catalogues of tool manufactures. A case consists of the problem description (i.e. a graphical description of the workpiece which has to be manufactured), and the problem solution (i.e. a production plan). A production plan is a sequence of operators by which the workpiece can be

[2]The integrated knowledge acquisition framework of the ARC-TEC project was developed by O. Kühn, F. Schmalhofer and G. Schmidt [Schmalhofer et al., 1991b].

produced from a mold. Each operator (called "cut" in mechanical engineering) consists of the cutting tool, the cutting parameters, and the cutting path.
For example, the cutting tool of operator M5-1 (see figure 1) is described by the string "CSSNL 3232 C15 SNGN151016 TO 3030", which is based on ISO norms for cutting tools in mechanical engineering. A character or a sequence of some characters at a certain position of the string represent a particular shape, size or style of the tool. For instance, the "C" at the beginning of the string specifies the insert clamping as "top clamping". The cutting operator is applied to remove a part of the workpiece with a cutting speed of $v_c = 450m/min$, a feed of $f = 0,45mm/rotation$ and a cutting depth of $d = 0,5mm$. The cutting material of the tool insert is abbreviated "SN 80" which is a specific type of "silicon-nitrite-ceramic".

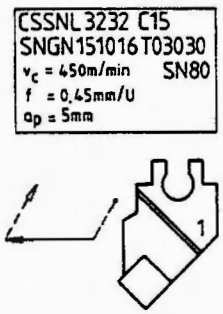


Figure 1: Mechanical engineering description of a cutting operator (M5-1)

In COKAM+ the graphical descriptions of complete cases which are scanned in from mechanical engineering catalogues are presented to the user. The pictures describe the context of the particular problems. They provide a very implicit and informal guidance. More formal knowledge about the cases is acquired in the integrated knowledge acquisition framework where the operators of the cases are classified in hierarchical classes. The hierarchical operator classes as input for COKAM+ are discussed in section 4.

2.2 The Model of Expertise

As described above, experts who develop production plans overcome the complexity of their task by doing modification planning with the help of cases (and sometimes in combination with "trial and error"-strategies).
With respect to this analysis and to the KADS methodology [Breuker and Wielinga, 1989, Wielinga et al., 1991] a very general problem solving model was developed by O. Kühn [Kühn et al., 1990] (also see [Schmalhofer et al., 1991b]). The most important aspect of the planning process is obviously the re-use of a generalized and good solution (skeletal plan, [Friedland and Iwasaki, 1985]) and its modification in order to get a new result (refinement).
This "skeletal plan refinement" - procedure [Köhler, 1991, Schmalhofer et al., 1991a] mainly described by a global KADS inference structure specifies the problem solving method that should be adopted by the desired knowledge based system. In the following this model is called *model of expertise.*

The global inference structure describes the problem solving process of production planning as follows: From the concrete description of the workpiece (product) and the available manufacturing environment more abstract feature descriptions are constructed. These abstractions are then associated with an appropriate skeletal plan that is available in the knowledge base. The skeletal plan is finally refined with the help of the workpiece and the factory description into the concrete production plan.
The metaclasses of the global inference structure specify six *views*: product, environment and plan, each detailed and general. We termed a metaclass *view* since it provides a certain view or filter of the acquired knowledge by the knowledge can be categorized. Relevant knowledge should fit at least one of the views. The model can therefore be utilized as a useful guidance for the knowledge acquisition framework as well as for judging the relevance of single knowledge units [3].

3 Integrated Knowledge Acquisition Framework

The knowledge is acquired from three different sources of information (texts, cases and the expert's respective memories) within the help of three tools: CECoS (Case Experience Combination System) [Bergmann and Schmalhofer, 1991, Tschaitschian, 1991], COKAM+ and SPGEN (Skeletal Plan Generation Procedure) [Schmalhofer et al., 1991a]. The three tools acquire the knowledge which is required by the model of expertise: abstract descriptions of the problem (operator class and problem class definitions), abstraction and refinement rules, and skeletal plans. Furthermore, the cases are involved in the knowledge acquisition process with the three tools. An overview of the integrated knowledge acquisition method is presented in figure 2.
The interaction between the three tools is determined by the integrated knowledge acquisition method. Considering each tool this integrated framework [Schmalhofer et al., 1991b] provides additional guidance. We want to concentrate on the interactions of COKAM+ with the other tools (see thick arrows in figure 2).

CECoS: First, the tool CECoS is applied. In this approach, CECoS is employed for delineating and defining hierarchies of operator classes from given concrete operators. CECoS first acquires similarity ratings of the cases from the expert and uses a cluster method to build an object hierarchy. Then, it elicits feature descriptions from the expert by showing him the hierarchy.
In [Bergmann and Schmalhofer, 1991] CECoS has been developed to support the delineation of a hierarchy of problem classes and their description by features. The problem classes are formalized stepwise whereby the formalization is guided by the model of expertise [Reinartz, 1991].

COKAM+: The interactive tool COKAM+ is applied to acquire formal descriptions of the operators in a production plan or operator classes hierarchy delineated by CECoS. It additionally acquires the appropriate task-related and common sense knowledge from text. Descriptions can be found in mechanical engineering textbooks, catalogues and documentations of a company. COKAM+ does not aim to comprehend the text completely,

[3] Meanwhile, a more detailed description of the model is developed [Kuehn and Schmalhofer, 1992] which is called "hierarchical skeletal plan refinement". This detailed model names possible combinations of the views and shows an elaboration of the "refinement" knowledge source. Since it is easier to directly refer to the six views and their combinations COKAM+ is based on the global model.

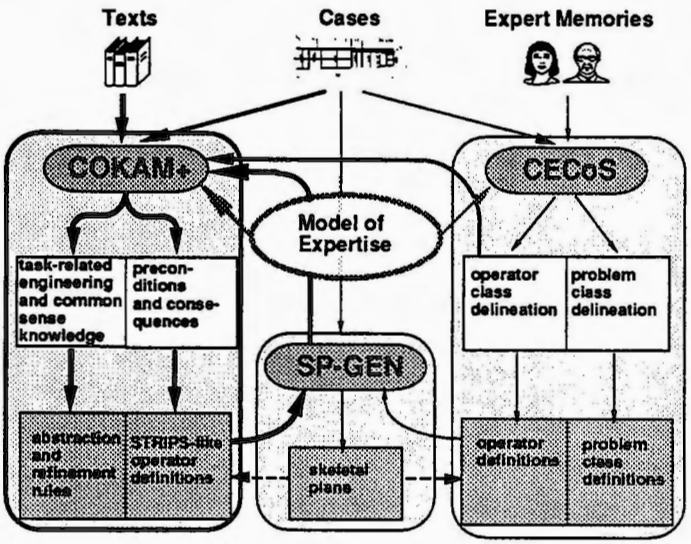


Figure 2: Integrated knowledge acquisition and involved tools

but to extract that knowledge from the text which is needed to obtain a sufficient *domain theory* as input for SPGEN (see the following sections).

SPGEN: SPGEN is based on explanation-based generalization as described in [Mitchell et al., 1986] in order to automatically construct skeletal plans. SPGEN obtains the production classes, acquired with CECoS, and a domain theory consisting of formal preconditions and consequences supplied by COKAM+. The domain theory is needed to explain the plan execution of a given, formalized case. By sequentially simulating each operator, the preconditions for its application are checked and its effects are determined. The effects of the operators are represented by a set of Strips–like rules with add- and delete-actions [Fikes and Nilsson, 1971, Nilsson, 1982]. Thus, SPGEN provides the requirement that the domain theory must contain a Strips–like description of the operators. If this domain theory is sufficient, a complete explanation of the plan will be obtained. Otherwise the domain theory is not complete because only such cases are selected which have been successfully used in a real world application. In other words, the simulation of the plan in SPGEN is a formal verification of the domain theory and provides a feedback concerning its completeness with respect to specific cases.
For the generalization process of SPGEN the domain theory must not only consist of the definition of concrete (or basic) operators (see M5-1 in figure 1) but it must also define classes of operators. Furthermore, a sequence of some operators, which the experts call macros, can be important for the construction of a skeletal plan.
Concerning the requirements of SPGEN, COKAM+ has to aim at a sufficient domain theory defining operators and their classes. And it must transfer the informal domain knowledge acquired from text in the Strips–like representation of SPGEN. In order to achieve this goal more efficiently CECoS supports COKAM+ by providing an operator class hierarchy. Further preparations of the operator classes are performed by the Formal Concept Analysis.

4 Definition of Operator Classes

4.1 Operator Hierarchy Defined by CECoS

The interactive knowledge acquisition tool CECoS is employed for defining hierarchies of operator classes from concrete operators. Since our real world application domain is very complex there are many categories of features by which operators can be classified. By the acquisition procedure of CECoS the important features can be extracted since the expert is allowed to give similarity ratings and he need not name features at the beginning. A hierarchy is calculated on the ratings by a cluster analysis. Now, the expert can name the important features corresponding to the classes of the hierarchy more easily. For a detailed description of CECoS see [Bergmann and Schmalhofer, 1991, Tschaitschian, 1991].
Figure 3 shows an example of an operator hierarchy with operators used to construct the two shafts M4 and M5. The vertical distance to the next higher node that two operators have in common indicates the similarity between them. For example, the two operators M5-7 and M4-6 are more similar than M5-1 and M4-1. This clustering is correct, since for both operators M5-7 and M4-6, the same cutting tools are used and only their cutting paths are slightly different, whereas M5-1 and M4-1 differ in the cutting path and cutting tools.

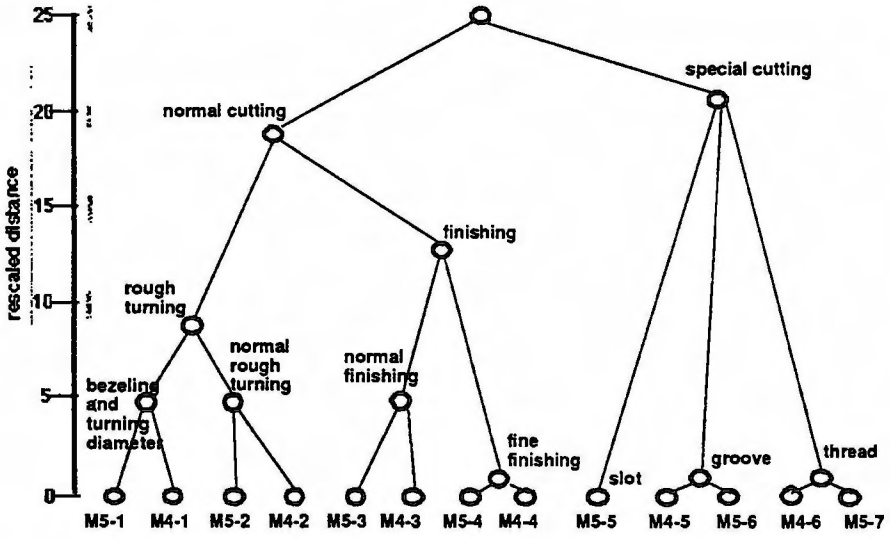


Figure 3: Hierarchical clustering of the operators of the cases M5 and M4

In higher levels classes are defined by associating terms (features), e.g. "bezeling and turning diameter" and "normal rough turning". These two classes can be generalized to the class "rough turning", as shown in the figure 3.
To acquire a domain theory with COKAM+ which SPGEN needs to generalize plans into skeletal plans, not only concrete operators and their classes, but also specific sequences of operators (often called macros in mechanical engineering) should be considered. In production plans, some specific types of operators often appear in the same order. Most of such sequences are well-known and the experts describe them by features, e.g. "processing of monotonic contour".

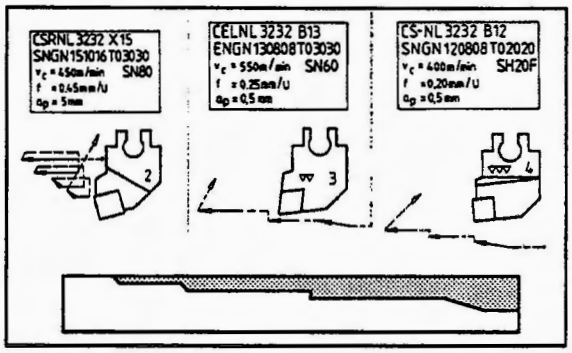


Figure 4: Sequence of operators processing a monotonic contour (M5-234)

Figure 4 shows a monotonic contour of a workpiece consisting of three shoulders and one bevel. The sequence to cut this contour starts with the normal rough cutting operator M5-2. Its cut, which is parallel to the rotational axis of the workpiece, is reapplied as many times as there are shoulders and bevels, i.e. it is replied for four times. Then, the "normal finishing" and the "fine finishing" are executed and their cutting paths always follow the contour.

4.2 Formal Concept Analysis

The operator hierarchy provided by CECoS may not be sufficient as an input for COKAM+. Besides definition of operator classes and their hierarchy improvements of representing the knowledge are investigated. The feature description of operators are suitable to apply a proved method for representing and handling conceptual hierarchies, the so called "Formal Concept Analysis" [Wille, 1982, Wille, 1987]. For instance this method enables an integrated representation of operators and sequences of operators (see figure 4) and supports the further formalization process of COKAM+ in an efficient way. Formal Concept Analysis is based on a set-theoretic model for conceptual hierarchies. This model mathematizes the philosophical understanding of a concept as a unit of thoughts consisting of two parts: the extension and the intention (comprehension); the extension covers all objects (or entities) belonging to the concept while the intention comprises all attributes (or features) valid for all those objects ([Wagner, 1973]). In the set-theoretic model we fix a set G, the elements of which are called *objects*, and a set M, the elements of which are called *attributes*; furthermore we assume a binary relation I between G and M where $(g, m) \in I$ (resp. gIm) is read: the object g *has* the attribute m. The triple (G, M, I) which is called a *(formal) context* is the basic structure of Formal Concept Analysis. For a context (G, M, I), the most frequently used (mathematical) operators are defined as follows:

$$\begin{aligned} A' &:= \{m \in M \mid gIm \text{ for all } g \in A\} \quad \text{for } A \subseteq G, \\ B' &:= \{g \in G \mid gIm \text{ for all } m \in B\} \quad \text{for } B \subseteq M. \end{aligned}$$

Now, a *(formal) concept* of a context (G, M, I) is defined to be a pair (A, B) with $A \subseteq G, B \subseteq M, A' = B$, and $B' = A$; A and B are called the *extent* and the *intent* of

the concept (A, B) respectively. The hierarchy of the concepts is given by the relation *subconcept- superconcept* which has to be defined for a context by $(A_1, B_1) \leq (A_2, B_2) :\Leftrightarrow A_1 \subseteq A_2 \ (\Leftrightarrow B_1 \supseteq B_2)$.
The set of all concepts of (G, M, I) with this order relation is a complete lattice called the *concept lattice* of (G, M, I). Thus, to any set S of concepts of (G, M, I) there always exists a largest subconcept, the infimum of S in the concept lattice, and a smallest superconcept, the supremum of S (for details see [Wille, 1982]).

	M5 -1	M5 -2	M5 -3	M5 -4	M5 -5	M5 -6	M5 -7	M4 -1	M4 -2	M4 -3	M4 -4	M4 -5	M4 -6	M5 -234	M4 -234
special cutting					×	×	×					×	×		
thread							×						×		
groove						×						×			
slot					×										
normal cutting	×	×	×	×				×	×	×	×			×	×
rough turning	×	×						×	×					×	×
finishing			×	×						×	×			×	×
bezeling and turning diameter	×							×							
normal rough turning		×												×	
normal rough turning for bevel									×						×
normal finishing			×											×	
normal finishing for bevel										×					×
fine finishing				×							×			×	×
monotone contour														×	
monotone contour + bevel															×

Figure 5: Context of the basic operators and sequences of operators

To demonstrate the basic notions and results of Formal Concept Analysis by an example we translate the operator hierarchy in figure 3 in a formal context and integrate two sequences of operators, M4-234 and M5-234, which is depicted in figure 4. This is now possible, since the tree structure of the hierarchy is extended to a (richer) lattice structure. The context is seen in figure 5; the objects are the basic operators and the mentioned sequences of operators. The attributes are the feature descriptions, e.g. "special cutting", and the binary relation is defined by "an operator has a feature". Additionally, there is a first inference given by the following inheritance: a sequence of operators has all features of its included basic operators in order to yield a hierarchical representation.
A concept of this context is for instance the pair consisting of the extent {M5-5, M5-6, M5-7, M4-5, M4-6} and the intent {special cutting}. All concepts of the context can be seen in figure 6 which shows the concept lattice by a line diagram (the bottom element ({ },M) is omitted, because it has no meaning in the context). A little circle labelled by a name of an object g represents $\gamma g := (\{g\}'', \{g\}')$, i.e. the smallest concept with g in its extent, and a little circle labelled by a name of an attribute m represents $\mu m := (\{m\}', \{m\}'')$, i.e. the greatest concept with m in its intent. The labelling allows to determine the extent and the intent for any concept (A, B) since $A = \{g \in G \mid \gamma g \leq (A, B)\}$ and $B = \{m \in M \mid \mu m \leq \gamma(A, B)\}$.
For instance

$$\gamma(\text{M5-6}) = (\{\text{M5-6, M4-5}\}, \{\text{groove, special cutting}\})$$

and the intent of the labelled concept "slot" is {slot, special cutting}, since "special cutting" lies above "slot". The equivalence $gIm \Leftrightarrow \gamma g \leq \mu m$ yields that the context can also be read from the diagram and the diagram contains all information of the context. This fact also implies that the hierarchy in figure 4 is entailed in the concept lattice.
Besides computing and representation of conceptual hierarchies the set-theoretical model based on attributes *and* objects enables a diversity of methods for data analysis. Inheritance, retrieval, realization, addition of new knowledge are some basic operations

in Formal Concept Analysis. They will be investigated to support the formalization of operators in COKAM+ and the applications of this formal knowledge.

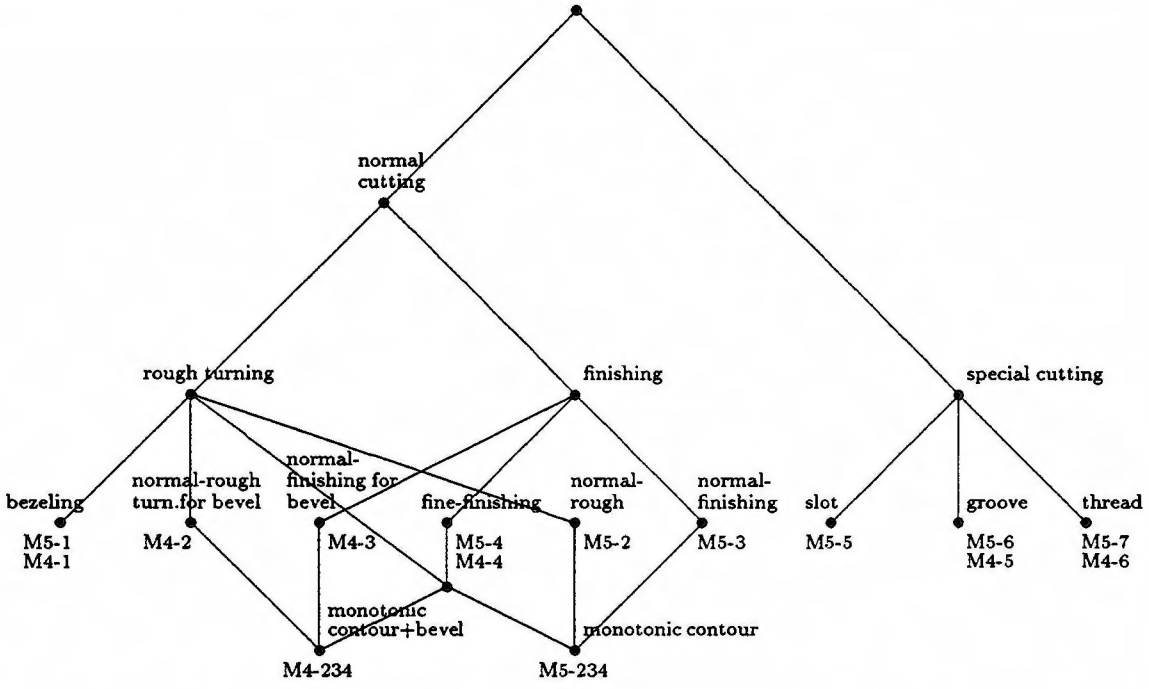


Figure 6: Concept lattice of the basic operators and sequences of operators

In general, representation, inference, acquisition, and communication of conceptual knowledge can be mathematically treated within the framework of Formal Concept Analysis [Wille, 1987, Luksch and Wille, 1991]. Different tools for data analysis are developed and tested in many applications [Ganter, 1987, Wille, 1989, Wille, 1992]. It may be efficient to investigate some of these tools for CECoS in order to obtain a conceptual hierarchy as an input knowledge structure for COKAM+ (see discussion in the last section).

5 Formalization of Operators

In this section, the model of expertise and the operator hierarchy serve as guidances for structuring and formalizing the knowledge elicited with COKAM+. COKAM+ expects the operator hierarchy as well as the features as input from CECoS and the lattice prepared by the Formal Concept Analysis.
Now, the knowledge which explains the selection of a specific operator must be acquired. In order to provide the correct input for SPGEN the explanations must be represented as preconditions or consequences of the operators or more detailed explanations of the preconditions.
COKAM+ extracts natural-language operator descriptions from texts as so called knowledge units. Such a knowledge unit consists of one natural language sentence containing one piece of information, e.g. one precondition for the application of an operator. The sentence of the knowledge unit is represented in one card of a Hypertext structure. By

extracting the knowledge unit from text, it has to be decontextualized so that it does not refer to the context of the text [Schmidt, 1992], e.g. anaphora have to be resolved.
All knowledge units are stored in an informal knowledge base which must be structured. For our application example about 70 knowledge units are extracted from several books. The selected knowledge units are numbered from 1 to 70. Table 1 lists some decontextualized knowledge units with their original number.

- "For ceramic cutting tools bezeling is required, if the mold has a rough surface." (**13**)
- "For rough turning the tool angle is 45° or 75°." (**29**)
- "The shank and the geometry of the insert of the selected tools must be appropriate so that they can cut the surfaces of the workpiece without collision." (**11**)
- "A large corner radius causes a better surface of the workpiece and stability of the insert." (**10**)
- "The new surfaces have a surface roughness of $R = \frac{f^2}{8r_\epsilon}$." (**33**)
- "The domain of application for silicon-nitrite-ceramic is the lathing and milling of cast iron workpieces with cutting speeds up to $800m/min$ or feeds up to $0.8mm/rotation$" (**7**)
- "Square inserts have the highest stability because of their large corner angle " (**9**)
- "SN 80 is a specific type of "silicon-nitrite-ceramic" (**68**)

Table 1: List of decontextualized knowledge units

5.1 Explanation of Basic Operators

For the explanation of basic operators each operator is shown to the expert. The expert then searches the informal knowledge base and selects all knowledge units which specify relevant preconditions and consequences of the particular operator. A graphical description of the mold, from which the workpiece has to be manufactured, the goal workpiece and the operators are graphically presented to the expert. Thus, the context of the concrete operator which must now be explained is presented to the expert. The expert links the knowledge units as preconditions, consequences or abstraction and refinement rules to this specific operator. If appropriate preconditions or consequences cannot be found in the informal knowledge base, the expert is to add new knowledge units. This procedure is the best way to really find all preconditions and consequences.
For table 1 such knowledge are selected which refer to the basic operator M5-1. The knowledge units number **13**, **29**, **11**, **7**, and **9** are preconditions of the operator. For example, knowledge unit number **13** explains that if the mold, from which the workpiece has to be manufactured, has a rough surface and if the cutting material is ceramic a preparing cut called "bezeling" has to be performed. This is a rather special precondition which only refers to a special cut like M5-1, whereas knowledge unit number **29** refers to all abstract operators which are "rough turnings". In the following, the preconditions are named by the letter "p" and their number.
Other knowledge units represent abstraction or refinement rules which are explanations of a precondition. The abstraction or refinement rules are applied to check whether the precondition can be fulfilled. In the table, such an abstraction rule is knowledge unit number **68** which explains that "SN80" is a special kind of "silicon-nitrite-ceramic"; this rule belongs to the precondition **7**. Henceforth, the abbreviation for such abstraction or refinement rules will be "ar" together with their number.

The last category of knowledge units are as consequences, e.g. "new surfaces" (**33**) and a "better surface of the workpiece and stability of the insert" (**10**) are consequences of M5-1. The formula of consequence number **33**, which is now called "c33", describes the surface roughness with the cutting feed f and the corner radius r_ϵ.
Comparing the explanation structures of two basic operators which are below a common node in the hierarchy, it can be seen that the operators have some preconditions and consequences in common. In the next section, it will be shown how this aspect can be used to define more general operators with respect to the operator hierarchy, i.e. the lattice structure.

5.2 Explanation of General Operators

Explanation structures are acquired for concrete operators as well as for generalized operators. The explanation hierarchy for preconditions and consequences are essential for the construction of more or less general skeletal plans.
In order to obtain the preconditions and consequences which describe general operators, they will be analyzed for basic operators. Then, the preconditions and consequences are related to a more general operator, if they referred to all operators which are subsumed by this general operator. Thereby, the abstraction and refinement rules are not taken into account since they belong to their precondition.
The set-theoretical model in the framework of Formal Concept Analysis provides a suitable solution for this task. At first, it enables a reduced analysis, since basic operators having the same features (and thus are not differentiated) are comprised. To obtain the abstraction for more general operators, i.e. more general concepts, the preconditions and consequences are added as new attributes to the hierarchic context in figure 5. The binary relation I is given by "a precondition refers to an operator" and "an operator has a consequence". Then, the inferences for sequences of operators are made. Each sequence of operators inherits all preconditions and consequences of its including basic operators. Note that these preconditions and consequences do not define the sequence as a whole. The resulting context of preconditions and consequences is shown in figure 7.
If we have a lattice structure as input hierarchy of which the basic operators build no anti-chain (which yields from the tree-hierarchy produced by CECoS) we may have a more reduced analyzing of the basic operators and more inferences caused by inheritance. Now, the abstraction to general operators is automatically given by concept building in the whole context, consisting of feature descriptions, preconditions, and consequences. These concepts are represented in the concept lattice seen in figure 8 (the bottom element is omitted again, since it has no meaning in this context).
If there are preconditions or consequences for all objects (basic operators) of a concept (more or less general operator), these preconditions or consequences are in the intent of the concept. For instance, "p9" and "c10" are in the intent of the concept labelled by "normal-rough" and in the intent of the concept labelled by "fine-finishing", because all objects (M4-1, M5-1, M4-2, M5-4, M4-4, M5-2) of these two concepts need precondition "p9" and have consequence "c10". Additionally, all preconditions and consequences of general operators are immediately given by the intent of the corresponding concept. For example the preconditions of the general operator "rough turning" are {rough turning}$''\cap${set of all preconditions} = {p5,p13,p17,p20,p22,p24,p25,p29}, and the consequences are {normal rough}$''\cap${set of all consequences} = {c31,c34,c35}.
One can see that many preconditions and consequences are labelled at more general operators like "normal cutting" or "finishing", in particular "p11-14, p16", and "c32" are at the top of the concept lattice. The higher a precondition is transferred in the hierarchy

	M5 -1	M5 -2	M5 -3	M5 -4	M5 -5	M5 -6	M5 -7	M4 -1	M4 -2	M4 -3	M4 -4	M4 -5	M4 -6	M5 -234	M4 -234
preconditions															
p1,p3,p7	×	×	×	×				×	×	×	×			×	×
p5,p13,p30	×							×							
p9	×	×		×				×	×		×			×	×
p17,20,22,24,25,29	×	×						×	×					×	×
p11-14,p16	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
p36-42			×	×						×	×			×	×
p45-54					×										
p57,p58						×						×			
p60-67							×						×		
consequences															
c10	×	×		×				×	×		×			×	×
c31	×							×							
c32	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
c33	×	×	×	×				×	×	×	×			×	×
c34	×	×						×	×					×	×
c35	×							×							
c43,c44			×	×						×	×			×	×
c55,c56					×										
c59						×						×			

Figure 7: Context of preconditions and consequences of basic operators

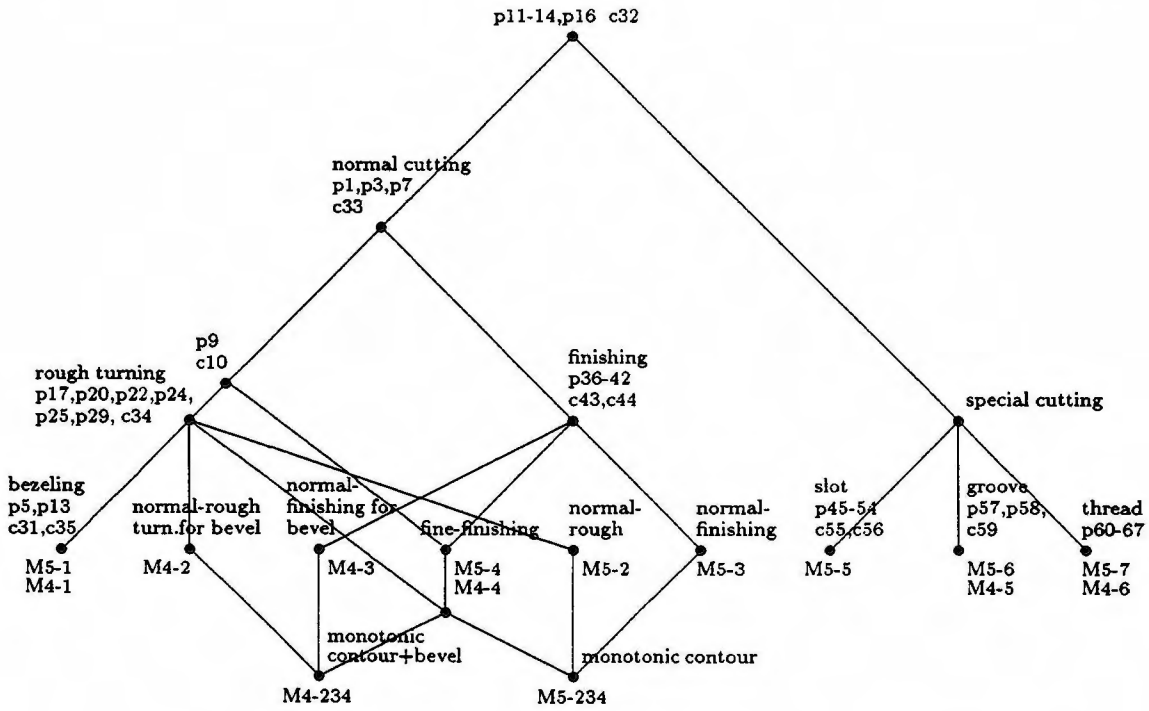


Figure 8: Concept lattice of feature description, preconditions and consequences

the sooner it influences the selection of the type of an operator and the smaller is its influence on the specific manufacturing process.
The result that most preconditions and consequences are referred to the general operators was expected since the knowledge units are extracted from textbooks. In textbooks the knowledge is more abstract and theoretic and seldom details of concrete operators are discussed.

5.3 Structuring With Respect to the Model of Expertise

The knowledge units, which are extracted natural language operator descriptions, are assigned to the views of the model of expertise to indicate how they will be used by the knowledge based system.
If a knowledge unit can not be assigned to at least one view its relevance for the system must be checked. The categorization of the knowledge units can be based on keywords which are related to the different views. For example, the precondition "p7" is assigned to the views general product, general environment and detailed plan. "Cast iron" (material of the workpiece) is a term referring to the product view. "Ceramic cutting tool" is a keyword which refers to the general environment view. Because of the concrete declaration of the type of insert clamping which fixes the insert of the cutting tool the knowledge unit is related to the detailed plan view, too.
There are some dependencies between structuring the knowledge units through the views and explaining the operators. For example, a precondition or a consequence is related to a (basic or generalized) operator or some of its parameters. The manufacturing plan consists of a sequence of operators. Therefore, a precondition or a consequence has to refer to the detailed or general plan view at least. This demand can be violated for three reasons. Either the categorization is wrong or the knowledge unit is not correctly used within the explanation structure or the knowledge unit itself is incomplete. By generating an appropriate template for formalizing the knowledge unit (see next section) such violations can be discovered. Then, the expert is allowed to change the knowledge unit, their categorization or their position in the explanation structure for removing the fault.

5.4 Knowledge Formalization

Now, the knowledge units are well prepared and can be transformed into a formal representation which can be utilized by the skeletal plan generation procedure SPGEN. The knowledge units are attached to the views of the model of expertise. Templates corresponding to a specific view or a combination of views are provided for the semi–formalization. Additionally, the templates in COKAM+ depend on whether the corresponding knowledge units are preconditions, consequences or refinement or abstraction rules. Several helpful restrictions can be defined because of this information. For example, preconditions are always related to an operator and therefore they have to be related to the views detailed or abstract plan. A very general template for a precondition can be defined:

$$precondition(operator, expression).$$

Considering all restrictions which can be defined for the operator definitions a simple context–free grammar can generate more specific templates for knowledge units. Terminal symbols of the grammar are given by the views and operator explanations associated with the knowledge units, e.g. general product as a view and precondition as a positions in the

explanation structure. Simple rewrite rules can then generate a template. For instance, as described above the precondition "p7" is related to the views general product, general environment and detailed plan. Thus, the corresponding template for this knowledge unit can be formulated in the following way:

$$\textit{"precondition(operator, expression((general product, general environment)} \rightarrow \textit{(detailed plan)))"}$$

The syntax of the formalization depends on the requirements of the tool which will apply the formal knowledge. Of course the syntax of templates can be adapted to this, too. The resulting formalization of the precondition "p7" which can be handled in SPGEN is:

$$\textit{"precondition("}M5-1\textit{",(workpiece_material(cast_iron), cutting_material(ceramics))} \rightarrow \textit{Feed} < 0.8, \textit{Speed} < 800)))\textit{"}$$

The preparation and the formalization of knowledge units are supported by COKAM+ which allows the construction of a child knowledge unit, i.e. a new card in the Hypertext structure, from any existing knowledge unit and records its source knowledge unit. Thereby, a new generation of knowledge units is constructed and maintained with the help of COKAM+. The established child–source–links between the knowledge units (for details see [Schmidt, 1992]) help to track down errors or inconsistencies which are later detected in the formal knowledge base. Since the acquisition is performed incrementally in small steps, such errors are also less likely to occur. Furthermore, these steps provide a good documentation of the process, e.g. for maintenance.

6 Application of the Formal Knowledge

There are two possible application scenarios of the knowledge acquired by COKAM+. Depending on the application scenario the knowledge should be interpreted in slightly different ways.

First application scenario: SPGEN The first possible application is the abstraction of case solutions to skeletal plans with the help of the domain theory acquired by COKAM+. The acquisition process of COKAM+ is especially constructed for this task within the integrated knowledge acquisition framework. It is focused on the operator definitions in a Strips-like notation [Fikes and Nilsson, 1971, Nilsson, 1982] and concentrates on the extraction and formalization of preconditions and consequences which are required by SPGEN within the integrated framework.
In SPGEN, the execution of a plan is simulated with the help of the domain theory. Thereby, for the single operators of the formalized case the formalized preconditions and consequences are checked whether they are fulfilled or not. Further, it is checked whether the acquired preconditions and consequences are sufficient to explain the operators completely. For this simulation process, specific types of preconditions and consequences are of interest. For example, only such preconditions which describe the relation of a current world state (concrete or general product or environment view) to a plan state (concrete or general plan view) are required. The preconditions which fulfill such conclusions can be selected and for their formalization specific templates can be defined.

For the abstraction of the concrete operators SPGEN can utilize the concept lattice of features descriptions, preconditions and consequences (see figure 8). Since in the simulation phase of SPGEN the concrete operators are explained the abstraction can start from these concepts. For example, the concrete operators M4-2 can be generalized to the more general operator "rough turning" by moving the lattice upwards. Another possible generalization is the recognition of a sequence of operators by moving the lattice downward. Here, the additional features concerning the sequence of operators must also be fulfilled. Regarding our example M4-2, it can also be checked whether there is a monotonic contour with a bevel so that the generalization to the macro operator M4-234 is possible.
This application scenario is involved in the knowledge acquisition process. The following scenario is an application task of the corresponding knowledge based system.

Second application scenario: Refinement of Skeletal Plans As described above the model of expertise specifies the problem solving process of the future knowledge based system. According to the model skeletal plans can be selected and must be refined to a concrete plan. This refinement process is the second application scenario for the knowledge acquired by COKAM+.
A skeletal plan can contain general operators in a specific order like: "bezeling", "normal rough turning", "normal finishing" and "fine finishing" with ceramic cutting tools. For the refinement of such a skeletal plan again the concept lattice in figure 8 is useful. It lists the possibilities which concrete operators can be selected and which preconditions must then be fulfilled.
Since there may be skeletal plans at different levels the refinement can also be performed at these different levels until a concrete plan is obtained. For example, the general operator "rough turning" can be refined to the less general operator "bezeling". We assume that for such a refinement process preconditions which were only assigned to the general plan view are needed. In the first application scenario such preconditions are not useful and by generating templates such an assignment will be detected as a fault. Perhaps, the assignment to the views can provide a categorization with respect to the specific applications, i.e. with respect to the special representations. This has to be investigated in our future works.

7 Discussion

Future Works: As seen in the sections above the set-theoretical approach of Formal Concept Analysis may be efficient in supporting knowledge formalization within COKAM+. The methods of Formal Concept Analysis can further be used for the application of the formal knowledge. Both the abstraction from case solutions to skeletal plans and the refinement process or the selection of concrete operators may be supported by the concept lattices and also by concrete tools like the so called object definition [Wille, 1987]. These methods have to be investigated and applied to concrete examples. It might be very successful to investigate the object definition for CECoS in order to integrate Formal Concept Analysis in both tools (CECoS and COKAM+) and their interactions.
In particular, an interactive tool to construct conceptual hierarchies of given objects (or attributes), the so called *object exploration* (*attribute exploration*) may be a very efficient method which can be used in CECoS. The tool object exploration makes the acquisition process of CECoS in one step. Since the similarity judgements might be also based on implicit feature descriptions, the object exploration is similar to the method used by

CECoS. But, it has some advantages as it is an exact, reduced, and complete procedure and results in a conceptual lattice structure.
Furthermore, we will have to elaborate and test the generation of templates and we plan to offer as much automatical support as possible for the filling of the templates with the respective domain knowledge. The formalization of the knowledge units will be further investigated, whether there are dependencies between the intended application scenario and the particular templates. In CommonKADS [Wielinga et al., 1992] recent knowledge modelling approaches are unified with respect to KADS [Wielinga et al., 1991] methodology. One new aspect of CommonKADS is that the domain knowledge will be more clearly structured: *"The type schemata that are required for a certain domain model depend on the nature of domain of application and on the reasoning process..."*. Our templates seem to be very similar to the domain model structure schemata in CommonKADS, since their generation is based on the views of the model of expertise, i.e. the reasoning process, and on the explanation structure of the cases, i.e. the application.

Summary: In this paper we show the acquisition of operator definitions in manufacturing with the tool COKAM+. Strong guidance from an integrated knowledge acquisition framework [Schmalhofer et al., 1991b], in which the tool is embedded, from the model of expertise and cases is needed to support the acquisition and formalization of the knowledge.
We briefly described possibilities to use a model as guidance for the development of adequate knowledge acquisition tools and their integration; this is performed in several, model-based knowledge acquisition approaches (to mention some: [Anjewierden et al., 1992, Shadbolt and Wielinga, 1990, Motta et al., 1990]). In our integrated knowledge acquisition framework the interaction with the other tools provide requirements which can be utilized for the acquisition process in COKAM+. On the other side, COKAM+ demands the operator hierarchy or the lattice prepared by the Formal Concept Analysis as input. In particular, it was analyzed whether a concept lattice can be acquired in our complex domain and whether this is more adequate than a hierarchy. Since COKAM+ expects that the hierarchy or the concept lattice is correct and rather complete it is based on prototypical cases and cannot be iteratively refined like in [Mitchell, 1982].
The KADS problem solving model of our application domain is skeletal plan refinement [Kühn et al., 1990]. PROTEGE I [Musen, 1989], which is a support tool for the knowledge engineer to construct knowledge acquisition tools, is also based on a skeletal plan refinement model [Linster and Musen, 1992]. However, this model also considers some time-oriented procedures. In both approaches the models provide strong guidance for the knowledge acquisition process with the expert. Our model structures the domain knowledge by its views and the acquired knowledge must fit in at least one of these categories. Furthermore, these views provide some help to generate templates for the formalization of the knowledge.
Contrary to most of the model-based approaches we explicitly require further guidance from the domain, i.e. guidance by the cases. This enhanced guidance can be a further useful support for transferring the informal knowledge into a formal representation.
With our paper we do not want to demand that the task-specific model-based approaches [McDermott, 1988, Wielinga et al., 1991] should become domain-specific. Such models are appropriate to obtain basic guidance which is helpful for the knowledge engineer. Furthermore, they are re-usable and can help to compare systems of different domains. But, if the acquisition tools should be used by the expert stronger guidance is necessary. Additional guidance can be obtained from cases [Schmalhofer and Thoben, 1992]. The knowledge engineer should consider this in the design of the tools.

Acknowledgements

We would like to thank Otto Kühn and Harald Gurres for their contributions to this work.

References

[Anjewierden et al., 1992] Anjewierden, A., jan Wielemaker, and Toussaint, C. (1992). Shelley - computer-aided knowledge engineering. *Knowledge Acquisition*, 4:109 – 125.

[Bergmann and Schmalhofer, 1991] Bergmann, R. and Schmalhofer, F. (1991). CECoS: A case experience combination system for knowledge acquisition for expert systems. *Behavior Research Methods, Instruments and Computers*, 23:142–148.

[Breuker and Wielinga, 1989] Breuker, J. and Wielinga, B. (1989). Models of expertise in knowledge acquisition. In Guida, G. and Tasso, C., editors, *Topics in Expert System Design, Methodologies and Tools*, Studies in Computer Science and Artificial Intelligence, pages 265 – 295. North Holland, Amsterdam.

[Chandrasekaran, 1988] Chandrasekaran, B. (1988). Gerneric tasks as building blocks for knowledge-based systems: the diagnosis and routine design examples. *The knowledge engineering review*, pages 183 –210.

[Eversheim, 1989] Eversheim, W. (1989). *Organisation in der Produktionstechnik, Arbeitsvorbereitung*, volume 4, pages 125–128. VDI Verlag, Düsseldorf.

[Fikes and Nilsson, 1971] Fikes, R. and Nilsson, N. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189 – 208.

[Friedland and Iwasaki, 1985] Friedland, P. E. and Iwasaki, Y. (1985). The concept and implementation of skeletal plans. *Journal of Automated Reasoning*, pages 161–208.

[Ganter, 1987] Ganter, B. (1987). Algorithmen zur formalen Begriffsanalyse. In Ganter, B., Wille, R., and Wolff, K. E., editors, *Beiträge zur Begriffsanalyse*, pages 241–254. B. I.-Wissenschaftsverlag, Mannheim.

[Köhler, 1991] Köhler, J. (1991). Approaches to the reuse of plan schemata in planning formalisms. Technical Memo TN-91-01, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI).

[Kuehn and Schmalhofer, 1992] Kuehn, O. and Schmalhofer, F. (1992). Hierarchical skeletal plan refinement task- and inference structures. In Bauer, C. and Karbach, W., editors, *Proceedings 2ndKADS User Meeting*, Munich. Siemens AG.

[Kühn et al., 1990] Kühn, O., Schmalhofer, F., and Schmidt, G. (1990). Diagnose von Wissenstypen fuer die Erstellung von Fertigungsplänen. Technical report, German Research Center for Artificial Intelligence. Interner Bericht des ARC-TEC-Projektes.

[Linster and Musen, 1992] Linster, M. and Musen, M. (1992). The inference structure of k-oncocin: Skeletal-plan refinement. In Bauer, C. and Karbach, W., editors, *Proceedings 2nd-KADS User Meeting*, Munich. Siemens AG.

[Luksch and Wille, 1991] Luksch, P. and Wille, R. (1991). A mathematical model for conceptual knowledge systems. In Book, H. H. and Ihm, P., editors, *Classification, Data Analysis and Knowledge Organisation*, pages 156–162. Springer-Verlag, Berlin-Heidelberg.

[McDermott, 1988] McDermott, J. (1988). Preliminary steps towards a taxonomy of problem-solving methods. In Marcus, S., editor, *Automating Knolwedge Acquisition for Expert Systems*, chapter 8, pages 225 –256. Kluwer Academic, Boston.

[Mitchell, 1982] Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, 18:203 – 226.

[Mitchell et al., 1986] Mitchell, T. M., Keller, R. M., and Kedar-Cabelli, S. T. (1986). Explanation-based generalization: A unifying view. *Machine Learning*, 1:47–80.

[Motta et al., 1990] Motta, E., Rajan, T., and Eisenstadt, M. (1990). Knowledge acquisition as a process of model refinement. *Knowledge Acquisition*, 2(1):21–49.

[Musen, 1989] Musen, M. (1989). Automated support for building and extending expert models. *Machine Learning*, 4(3,4):347–376.

[Nilsson, 1982] Nilsson, N. J. (1982). *Principles of Artificial Intelligence.* Springer, Berlin, Heidelberg.

[Reinartz, 1991] Reinartz, T. (1991). Definition von Problemklassen im Maschinenbau als eine Begriffsbildungsaufgabe. DFKI-Document D-91-18, German Research Center for Artificial Intelligence (DFKI), Kaiserslautern, Germany.

[Schmalhofer et al., 1991a] Schmalhofer, F., Bergmann, R., Kühn, O., and Schmidt, G. (1991a). Using integrated knowledge acquisition to prepare sophisticated expert plans for their re-use in novel situations. In Christaller, T., editor, *GWAI-91 15th German Workshop on Artificial Intelligence*, pages 62 – 71. Springer-Verlag.

[Schmalhofer et al., 1991b] Schmalhofer, F., Kühn, O., and Schmidt, G. (1991b). Integrated knowledge acquisition from text, previously solved cases and expert memories and expert memories. *Applied Artificial Intelligence*, 5:311 – 337.

[Schmalhofer and Schmidt, 1991] Schmalhofer, F. and Schmidt, G. (1991). Situated text-analysis with COKAM+. In *European Knowledge Acquisition Workshop, Sisyphus Working Papers : Text Analysis.* J. Boose, and B. Gaines and M. Linster and D. Smeed and B. Woodward.

[Schmalhofer and Thoben, 1992] Schmalhofer, F. and Thoben, J. (1992). The model-based construction of a case-orieneted expert system. *AI Communications*, 5(1):3 – 18.

[Schmidt, 1992] Schmidt, G. (1992). Knowledge acquisition form text in a complex domain. In Belli, F. and Radermacher, F., editors, *Industruial and Engineering Applications of Artificial Intelligence and Expert Systems*, Lecture Notes in Artificial Intelligence, pages 529 –538. Springer Verlag. 5th International Conference, IEA/AIE-92, Paderborn, Germany.

[Schmidt et al., 1990] Schmidt, G., Legleitner, R., and Schmalhofer, F. (1990). Lautes Denken bei der Erstellung der Schnittaufteilung, der Werkzeugauswahl und Festlegung der Maschineneinstelldaten. Diskussionspapier 90/13, DFKI, Kaiserslautern, Erwin-Schrödinger-Str. (Bau 57), 6750 Kaiserslautern.

[Schmidt and Schmalhofer, 1990] Schmidt, G. and Schmalhofer, F. (1990). Case-oriented knowledge acquisition from texts. In Wielinga, B., Boose, J., Gaines, B., Schreiber, G., and van Someren, M., editors, *Current Trends in Knowledge Acquisition*, pages 302–312, Amsterdam. IOS Press.

[Shadbolt and Wielinga, 1990] Shadbolt, N. and Wielinga, B. (1990). Knowledge based knowledge acquisition: the next generation of support tools. In Wielinga, B., Boose, J., Gaines, B., Schreiber, G., and van Someren, M., editors, *Current trends in knowledge acquisition*, pages 313–338, Amsterdam. IOS.

[Spur, 1979] Spur, G. (1979). *Produktionstechnik im Wandel.* Carl Hanser Verlag , München.

[Thien and Chien, 1985] Thien and Chien (1985). *An introduction to automated process planning systems.* Prentice-Hall.

[Tschaitschian, 1991] Tschaitschian, B. (1991). Eine integrative Wissenserhebung mit CECoS: Konzepte und prototypische Implementierung. Master's thesis, University of Kaiserslautern. Studienarbeit im FB Informatik.

[Wagner, 1973] Wagner, H. (1973). Begriff. In *Handbuch philosophischer Grundbegriffe*, pages 191–209. Kösel, München.

[Wielinga et al., 1991] Wielinga, B., Schreiber, A. T., and Breuker, J. A. (1991). KADS: A modelling approach to knowledge engineering. Esprit Project P5248 KADS-II KADS-II/t1.1/PP/UvA/008/1.0, University of Amsterdam.

[Wielinga et al., 1992] Wielinga, B., van de Velde, W., Schreiber, G., and Akkermans, H. (1992). Towards a unification of knowledge modelling approaches. Esprit Project P5248 KADS-II: KADS-II/T1.1/UvA/RR/004/2.0.

[Wille, 1982] Wille, R. (1982). Restructuring lattice theory: an approach based on hierarchies of concepts. In Rival, I., editor, *Ordered Sets*, pages 445–470, Dodrecht-Boston.

[Wille, 1987] Wille, R. (1987). Bedeutung von Begriffsverbänden. In Ganter, B., Wille, R., and Wolff, K. E., editors, *Beiträge zur Begriffsanalyse*, pages 161–211. B.I.-Wissenschaftsverlag, Mannheim.

[Wille, 1989] Wille, R. (1989). Knowledge acquisition by methods of formal concept analysis. In Diday, E., editor, *Data Analysis, Learning Symbolic and Numeric Knowledge*, pages 365–380. Nova Science Publishers, New York - Budapest.

[Wille, 1992] Wille, R. (1992). Concept lattices and conceptual knowledge systems. *Computers and Mathematics with Applications*, (23):493–515.

REPGRID-NET: COMBINING CONCEPTUAL MODELING WITH ELECTRONIC MAIL TO PROVIDE GROUP DECISION SUPPORT

M L G Shaw & B R Gaines

Knowledge Science Institute, University of Calgary
Calgary, Alberta, Canada T2N 1N4
(mildred@cpsc.ucalgary.ca, gaines@cpsc.ucalgary.ca)

Abstract: This paper reports an attempt to move computer-based techniques for supporting the analysis of group cognitive processes and decision-making from being specialist applications to becoming a routine organizational tool used as readily as electronic mail. The objective is to support the discourse processes of functional groups within an organization by enabling them to investigate, analyze and compare the conceptual frameworks of those playing roles within the group. In particular, the system developed shows when individuals are in conflict through using the same term for different concepts, or in tacit correspondence through using different terms for the same concept. It provides support for group knowledge and decision processes as an integrated extension to electronic mail requiring no particular expertise in use or supervision. The work reported is part of a larger study of knowledge support systems merging artificial intelligence and computer communication techniques.

INTRODUCTION

There has been an interesting historical evolution in the organizational application of conceptual model tools based on personal construct psychology. These tools were developed by Kelly (1955) for clinical applications in the 1950s, became widely used in social, educational and organizational psychology in the 1960s, were incorporated in interactive computer programs in the 1970s (Shaw, 1980), and adopted as knowledge acquisition tools for knowledge-based system development in the 1980s (Shaw & Gaines, 1983; Boose, 1984). The knowledge acquisition research community has led to major advances in the interactive elicitation and on-line analysis of conceptual models, but the focus has been on the elicitation of expertise and its transfer to knowledge-based systems (Boose & Gaines, 1988, 1990; Gaines & Boose, 1988; Boose, 1989). Many of the significant early applications of personal construct psychology to organizational modeling have been neglected, even though the advances in the technology have opened up new opportunities for effective application.

In particular, in knowledge acquisition research, a significant growth area has been the support of group processes through sharing and comparison of personal knowledge in group decision support contexts. Generic toolboxes encompassing and integrating a range of methodologies such as KSSO (Shaw & Gaines, 1987; Gaines, 1988) and AQUINAS (Boose & Bradshaw, 1987), what have come to be called second-generation knowledge acquisition tools (Gaines & Linster, 1990), already make provision for users with widely varying roles, requirements and capabilities. Liou, Weber and Nunamaker (1990) have reported research on a methodology for knowledge acquisition in a group decision support system environment. Methodologies have also been developed that manage the elicitation of knowledge from groups of experts and are supported by tools that can analyze relations between the conceptual frameworks, and conflicts in the use of terminologies, for groups comprising multiple experts and clients (Gaines & Shaw, 1989; Shaw & Gaines, 1989).

However, all the research and applications to date have focused on the use of conceptual modeling tools in specific, planned studies, rather than as continuously available

organizational tools such as email and conferencing facilities. This seems to reflect both the past state of the art in information technology and the resultant attitudes to realistic system development. We have moved from an era of batch processing to one of highly interactive computing through timesharing and personal computing, but our applications software is still targeted on supporting individual rather than group processes. As local and global electronic networks have come into everyday use within organizations interest has increased in research on systems developed to specifically support collaborative activities. It is important to remember that much significant technology already exists, and to examine its re-development within a group decision support environment.

The studies reported in this paper are part of an attempt to move computer-based techniques for supporting the analysis of group cognitive processes and decision-making from being specialist applications, typically used once in studying or supporting a group, to becoming a routine organizational tool used as readily as electronic mail. We have taken electronic mail as our starting point for a number of reasons. First, it is already widely used within organizations on a routine basis (Kerr & Hiltz, 1982; Quartermain, 1990). A logical, and psychologically harmonious, extension to email is far more likely to be accepted and used routinely than a new application where a major decision has to be made about use. Second, we see human discourse as an essential process in the formation of functional groups within society that is supported at a very basic level by email. Our objective is to extend the support of discourse through tools that make conceptual models more overt as an aid to mutual understanding—what Habermas (1981) has termed "communicative action."

This paper reports some preliminary experiments on the integration of personal construct psychology elicitation and analysis tools with an electronic mail system to support functional groups within an organization, and the structured development and exchange of knowledge within such groups.

UNDERLYING COGNITIVE AND SOCIAL THEORY

With the power of current technology it is easy to create impressive systems that are not psychologically and socially functional because they do not mesh with human socio-cognitive processes. On the other hand, there is no widely agreed model of the underlying dynamics of such processes—cognitive and organizational psychology research is an area of explosive growth undergoing as rapid a change as computing technology (Gaines, 1991), particularly as our understanding of the modern era is enhanced by post-modern perspectives (Heller, 1990). This section outlines the theoretical perspectives underlying the system development reported in this paper.

There are three basic premises underlying our work: that knowledge processes in society can be understood as the modeling procedures of a distributed anticipatory system; that these modeling procedures can themselves be modeled in terms of the distinctions made; and that the cognitive processes of individuals and organizations can be understood within this framework (Gaines, 1989). Simmel made this inter-relation of wholes and parts the center piece of his sociology:

> "Society strives to be a whole, an organic unit of which the individuals must be mere members. Society asks of the individual that he employ all his strength in the service of the special functions which he has to exercise as a member of it; that he so modify himself as to become the most suitable vehicle for this function....man has the capacity to decompose himself into parts and to feel any one of these as his proper self." (Simmel, 1950, pp.58-59)

Simmel's insight that the group member is always a fragment of a person, a role created precisely to enable the person to enter the group has been developed extensively by Wolff with his notions of *surrender* and *catch*:

"From the standpoint of the world of everyday life, the mathematician, as we often put it, lives in the 'world of mathematics', dealing with 'nonreal' elements, notably numbers, whose relation to 'real' things, to 'reality', is not part of his concern. Analogously for the logician. What makes our subject-object approach to this attitude misleading is the fact that the subject, the student of mathematics or logic—his or her individuality, including motives and attitudes—is irrelevant for our understanding; the only thing that counts is the pursuit, with its results and questions." (Wolff 1976 pp.162-163)

He makes the key point that not only does the real world, the object, disappear to be replaced by the world of mathematics, but also that in entering into this world the person doing mathematics, the subject, also disappears to be replaced by a new entity, the mathematician.

Wolff was not alone in these insights and there are two further models which develop and complement his. In terms of his "subject" Pask's (1975) concept of "P-individuals" as coherent psychological processes capable of engaging in conversations is a useful representation of the results of Wolff's "surrender", particularly when we note that several P-individuals may execute within a single processor. Thus 'the mathematician' may engage in conversation with 'the physicist' or 'the statesman' all of whom happen to use the same brain for their processing. In terms of Wolff's "object", corresponding to our notions of "knowledge", Popper's (1968) concept of a "third world" of "statements in themselves" is a useful representation of that which we "catch", particularly when we note its distinct ontological status:

"I regard the third world as being essentially the product of the human mind. It is we who create third-world objects. That these objects have their own inherent or autonomous laws which create unintended and unforeseeable consequences is only one instance (although a very interesting one) of a more general rule, the rule that all our actions have such consequences." (Popper, 1974 p.148)

Pask goes beyond Simmel in conceiving that a P-Individual is not just what we conventionally term a 'role' within a person (which he terms an M-Individual, or mechanically characterized individual) but may itself be composed of a number of roles coming together to form a unity that we conventionally term a group or organization:

"a P-Individual...has many of the properties ascribed by anthropologists to a role, in society or industry, for example. A P-Individual is also a procedure and, as such, is run or executed in some M-Individual, qua processor. However, it is quite exceptional to discover the (usually assumed) one to one correspondence between M-Individuals and P-Individuals." Pask (1975 p.302)

Shaw (1985) has developed Pask's notions within the framework of personal construct psychology to show how Kelly's cognitive psychology may be used to account for the psychological processes not only of individual people but also for that of functional groups such as a nuclear family or a product executive. Figure 1 illustrates these ideas. Anne is shown as having three roles, wife, mathematician and technical vice president. John is also shown as having three roles, husband, golfer and sales vice president. The wife role in Anne together with the husband role in John together constitute a cognitive entity, a nuclear family. This has behavior, language, legal rights, and a model of the world, that are distinct from those of Anne and John in their other roles, and goes beyond those of the participating roles in Anne or John alone. Similarly, the technical VP role in Anne, the sales VP role in John, the finance VP role in Mark and the production VP role in Sue constitute another cognitive entity with again behavior, language, rights and models that are distinct from its participants and other cognitive entities in which they participate in other roles.

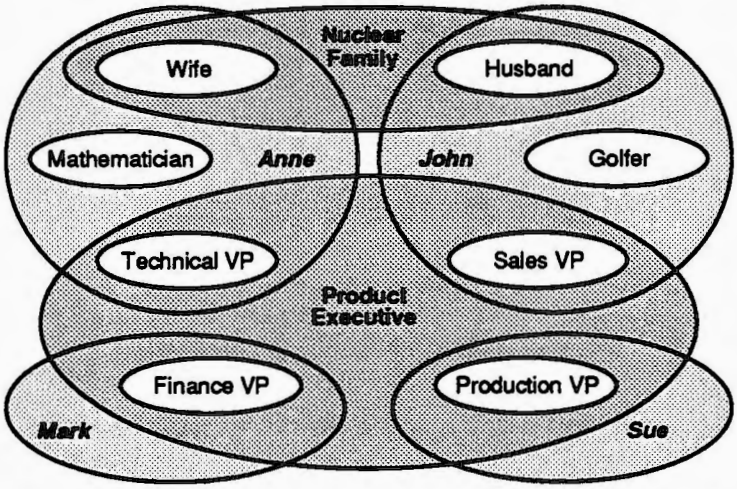


Fig. 1 Roles and groups as psychological individuals

Switching from socio-cognitive theory to computer technology, we can see email as providing a communication channel for discourse between the individuals involved. However, it provides no structures other than linguistic communication for supporting the cognitive processes underlying that discourse. For example, how do the concepts and terms used by the technical VP in discussing the rationale for a new product relate to those of the Sales VP? They may be using the same term for different concepts. They may use different terms for the same concept. They may have terms and concepts in their particular roles that are part of what that role peculiarly contributes to the group cognitive entity, and are not understood by the others involved. In an offline context, there are interactive computer-based tools specifically designed to elicit, analyze and present such conflicts, correspondences and contrasts in a group decision support framework (Shaw & Gaines, 1989). However, as noted above, they are used for specific 'studies', not as a routine part of the discourse and decision process. It is interesting to investigate what will happen if such tools are removed from their specialist isolation and integrated into routine organizational communication.

The theoretical foundations for developing eliciting, analyzing and comparing the conceptual frameworks of the cognitive entities in Figure 1 is derived from the notion that concepts correspond to terms applied to *distinctions* that we make in modeling a world:

> "a universe comes into being when a space is severed or taken apart...By tracing the way we represent such a severance, we can begin to reconstruct, with an accuracy and coverage that appear almost uncanny, the basic forms underlying linguistic, mathematical, physical and biological science, and can begin to see how the familiar laws of our own experience follow inexorably from the original act of severance." (Brown 1969)

Personal construct psychology notes that distinctions used by people often occur as a disjoint pair of concepts termed a *construct,* and that Brown's program of reconstruction is carried out using a technique termed the *repertory grid* to investigate the constructs used by an individual in a role (Shaw, 1980).

ELICITING AND COMPARING CONCEPTUAL MODELS

Repertory grid elicitation is an effective method for eliciting conceptual models in a domain (Shaw, 1980; Gaines & Shaw, 1980), and has been widely used in management, education, clinical psychology and the development of knowledge-based systems (Shaw & Gaines, 1983, 1987; Boose, 1984, 1986). It is an extensional approach in which

individuals are asked to specify a set of entities in a domain, then make distinctions among them, naming the distinctions and classifying all the specified entities in terms of them. The distinction determined in this way is an approximation to the underlying concept since critical entities may be missing. However, computer-based elicitation techniques attempt to prompt the individual for additional entities to discriminate between extensionally related distinctions (that is, making the same, or similar, classifications). Group comparisons, as discussed in this paper, have similar dynamics—an extensionally apparent correspondence may be accepted or rejected, and the rejection may be based on the specification of additional entities as counter-examples.

The set of distinctions made by a person in a role, and the relations between them, characterizes the conceptual models used by the person in that role. In comparing conceptual models between different roles and different people, there are two significant dimensions of comparison: one of the distinctions themselves, do they use equivalent distinctions?; and one of terminology, how are the distinctions named? Figure 2 shows how these dimensions combine to give four different relations between concepts: *consensus* arises if the conceptual systems assign the same term to the same distinction; *conflict* arises if the conceptual systems assign the same term to different distinctions; *correspondence* arises if the conceptual systems assign different terms to the same distinction; *contrast* arises if the conceptual systems assign different terms to different distinctions.

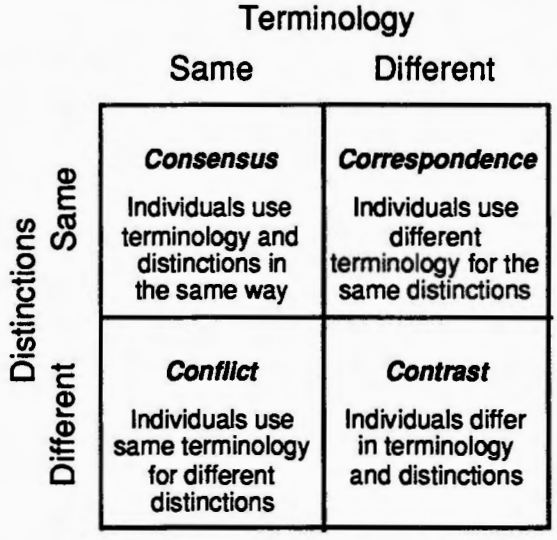


Fig. 2 Consensus, correspondence, conflict and contrast

The recognition of consensual concepts is important because it establishes a basis for communication using shared concepts and terminologies. The recognition of conflicting concepts is important because it establishes a basis for avoiding confusion over the labeling of differing concepts with same term. The recognition of corresponding concepts is important because it establishes a basis for mutual understanding of differing terms through the availability of common concepts. The recognition of contrasting concepts is important because it establishes that there are aspects of the differing expertise about which communication and understanding may be very difficult, even though this should not lead to confusion. Such contrasts are more common than is generally realized. For example, it is possible to derive the same theorem in mathematics either by using an algebraic perspective, or a geometric one. There is nothing in common in these two approaches except the final result. It may still be possible to discuss the same domain using consensual and corresponding concepts that were not fundamental to the problem solving activities.

We have previously developed computer-based interactive tools for the elicitation, analysis and comparison of conceptual structures using repertory grid techniques (Shaw, 1980; Gaines & Shaw, 1989; Shaw & Gaines, 1989). *RepGrid* is a knowledge support system providing an integrated set of tools for elicitation and analysis of elements and constructs in a given domain. As shown in Figure 3, it combines a number of different techniques, including element and construct elicitation and clustering, and is linked to an inductive rule generation program. It runs on the Apple Macintosh family of computers to provide a highly interactive and graphic knowledge acquisition environment. At the heart of RepGrid is an object-oriented knowledge base in which knowledge is formally represented as a multiple-inheritance digraph of classes, objects and properties.

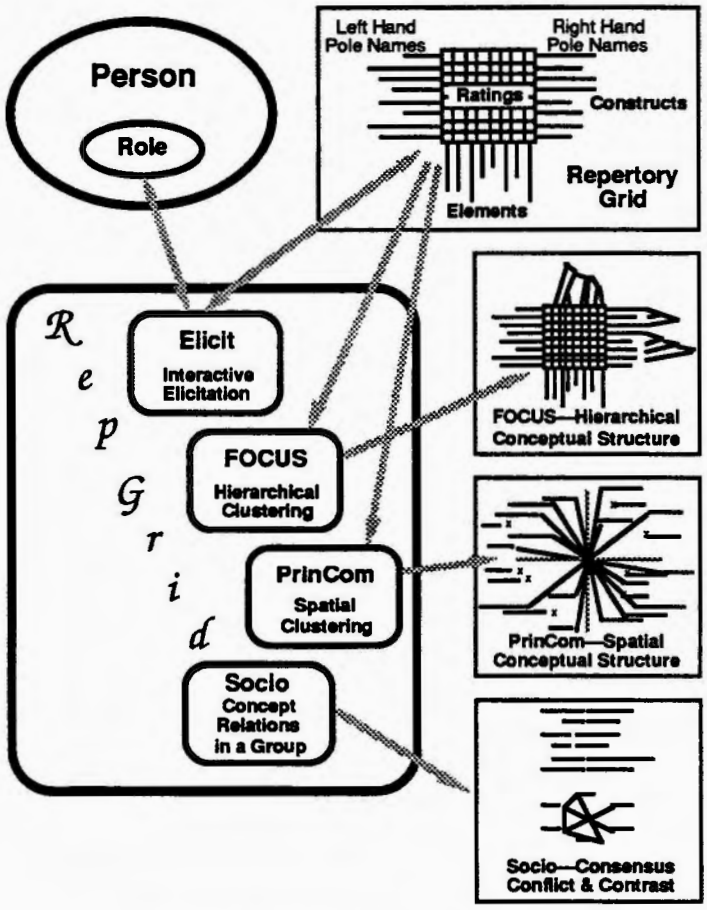


Figure 3 RepGrid Functionality

The main tools in RepGrid are shown in Figure 3:

- *Elicit* accepts specifications of elements within a domain and provides an interactive graphical elicitation environment within which a person can distinguish elements to derive his or her constructs within the domain. The resultant conceptual system is continuously analyzed to provide feedback prompting the person to enter further elements and constructs.
- *Exchange* extends this to share elements and constructs between people and allows the terms in the conceptual system derived from one person to be used by another in order to determine whether the two conceptual systems are different in any way. It can also be used by the same person looking at changes in their own conceptual structures over time, for example, after reading a specific book, or exploring a particular domain.

- *Process* gives access to clustering tools for the analysis and display of the conceptual systems elicited: FOCUS shows the system as a hierarchical structure; and PrinCom as a spatial map.
- *Socio* processes results from several people to reveal the similarities and differences in their conceptual systems, or the same person at different times, construing a domain defined through common elements or constructs. It can be used to focus discussion between people on those differences between them which require resolution, enabling them to classify them in terms of differing terminologies, levels of abstraction, disagreements, misunderstandings, and so on.

RepGrid is coupled through an inter-application protocol with Apple's HyperCard, and this has been used to integrate it with an email system to develop RepGrid-Net as described in the following section.

REPGRID-NET: A NETWORKING SUPPORT SYSTEM

RepGrid-Net is a computer-based message system that integrates conventional electronic mail and bulletin board facilities with the repertory grid elicitation and analysis facilities described above to provide both unstructured and structured communications supporting the formation and operation of special interest networks. Users see a mail system in which special-interest groups are specifically supported.

The coordinator of a special-interest group provides a basic focus for it through statements of intent, topics and issues which are handled on a bulletin board basis. He or she also provides one or more kernel grids listing specific topics and the concepts which they apply to them. These kernel grids can be developed by others interested in the groups, using the stated topics and concepts, and adding to them.

General similarities between grids are analyzed to provide a socionet of people with common viewpoints, and this may be used to access the mail system to communicate with them. Detailed comparisons of similarities and differences between viewpoints may be made, and individual concept structures can be analyzed. The elicitation and analysis facilities are all highly graphical, designed to be easy and interesting to use.

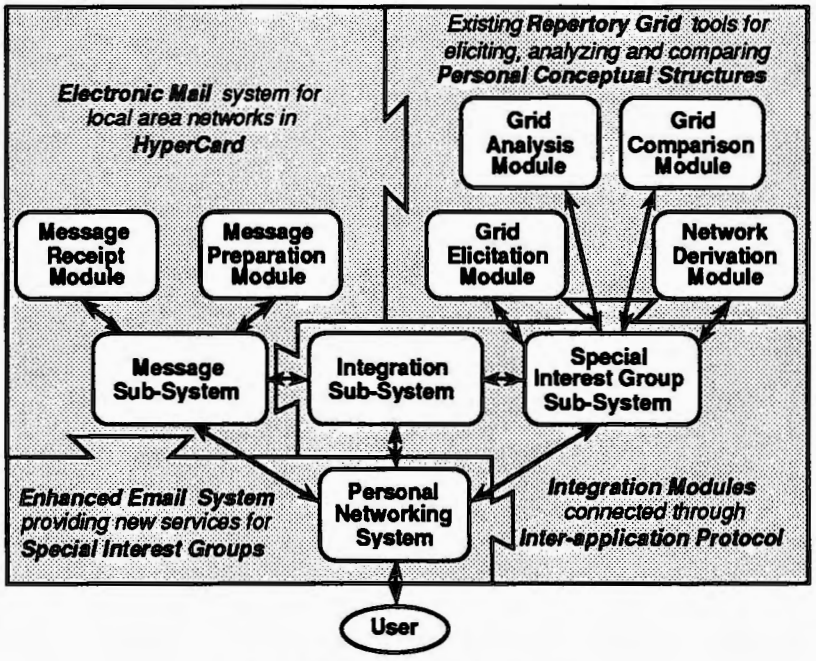


Fig.4 System architecture

Figure 4 shows the overall systems architecture. The message sub-system is written in HyperCard and is conventional in its operation. Users register with the system and may access lists of those with mailboxes registered on the system, lists of incoming mail and confirmations that outgoing mail has been accessed (see Figure 9). They may prepare and send mail, and receive and reply to mail. The special interest group sub-system is described in detail in the next section. It handles the group bulletin board and grid elicitation and analysis. The integration sub-system manages the interactions between the other two sub-systems, for example, that the graphic displays of networks of people with common interests may be used to send a message to one or more of them.

RepGrid-Net is implemented on a network of Macintosh computers coordinated through AppleTalk access to an AppleServe file server. Users see a conventional Macintosh program interface at all times and, even though two distinct programs are running, they communicate through inter-application protocols in such a way that there appears to be only a single application.

SYSTEM OPERATION

This section presents RepGrid-Net in operation for a product development group concerned with microwave ovens. An assistant to the chief executive has already entered a kernel grid on "Project67" concerned with relevant existing and competitive products, and with the major new product proposals under consideration. The participants are Anne, John, Sue and Mark in their roles as shown in Figure 1. We will go through a sequence of Anne's interaction with the system. She sees the screen of Figure 5 at login.

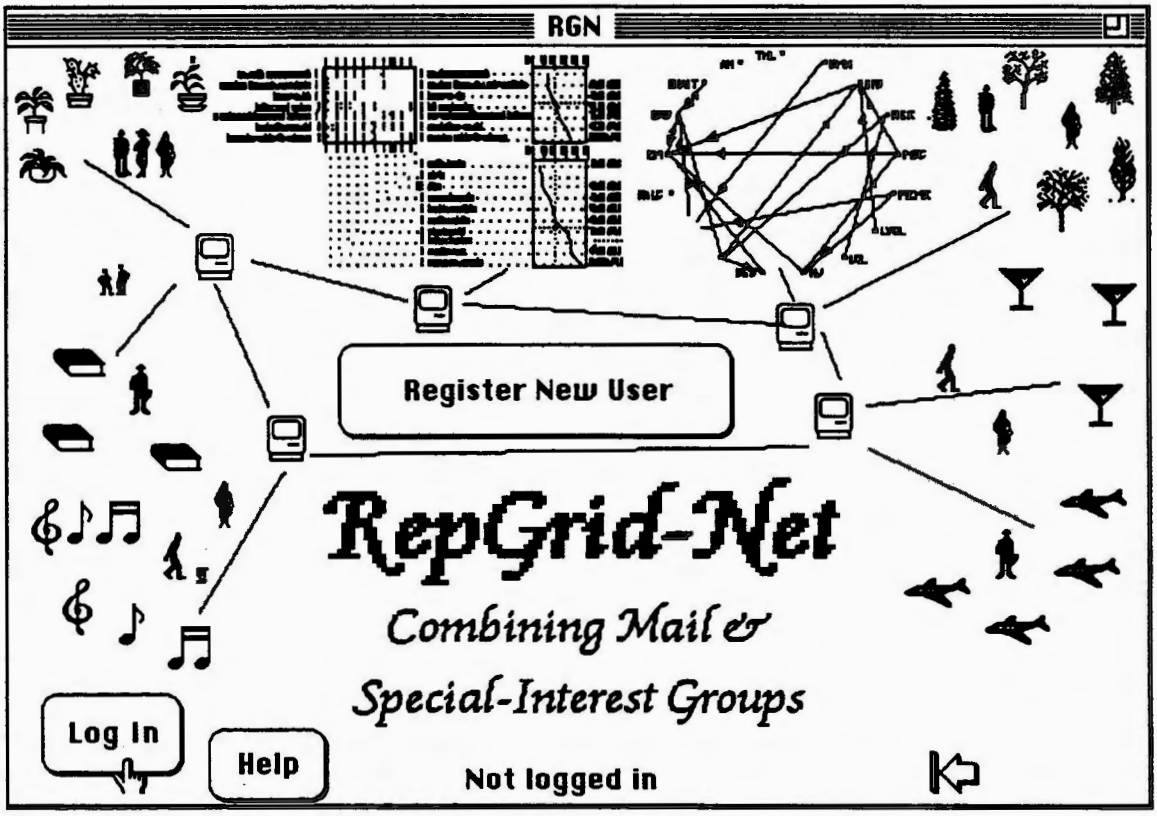


Fig.5 Initial login screen

Anne selects "Log In" and goes to the screen shown in Figure 6.

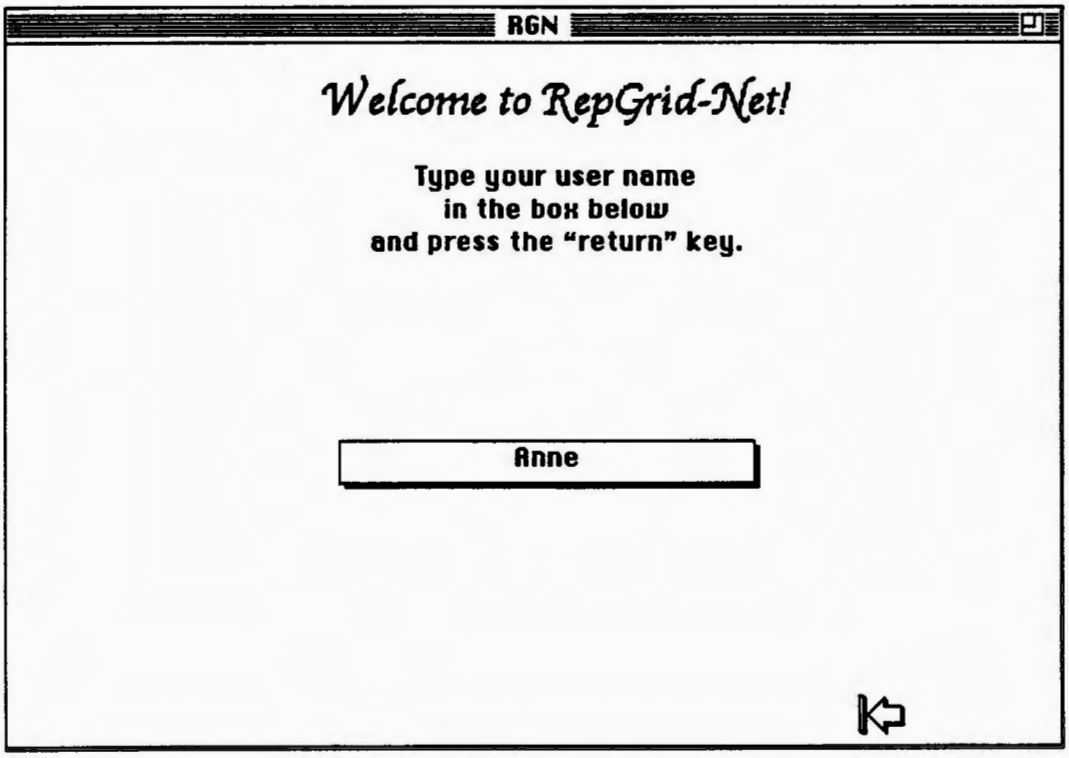


Fig.6 Log in and registration screen

She types in her name, logs in and goes to the screen shown in Figure 7.

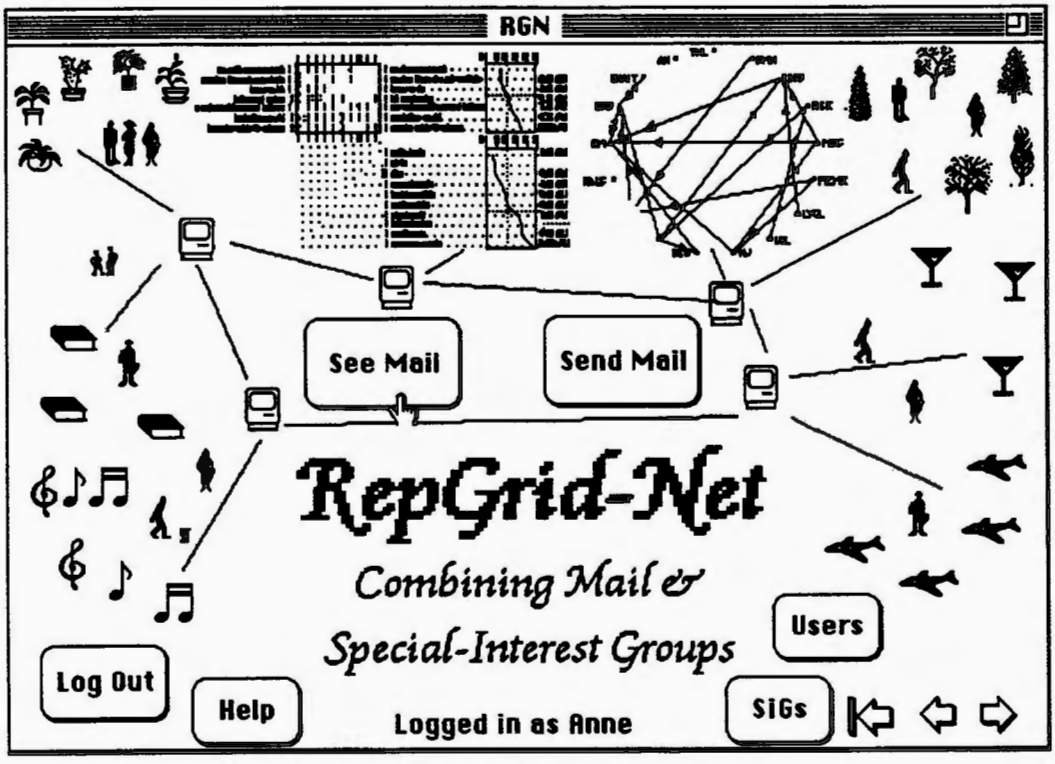


Fig.7 Main access to functionality screen

Anne selects "See Mail" and goes to the screen shown in Figure 8.

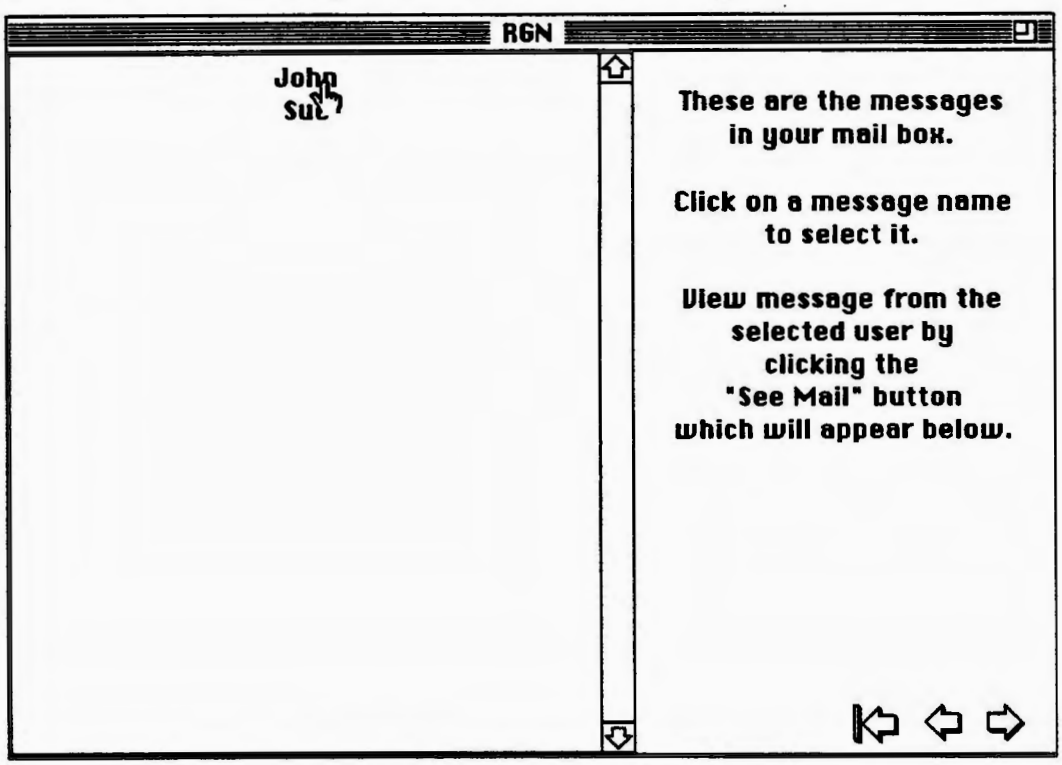


Fig.8 List of incoming mail screen

She clicks on "John" to access the mail and goes to the screen shown in Figure 9.

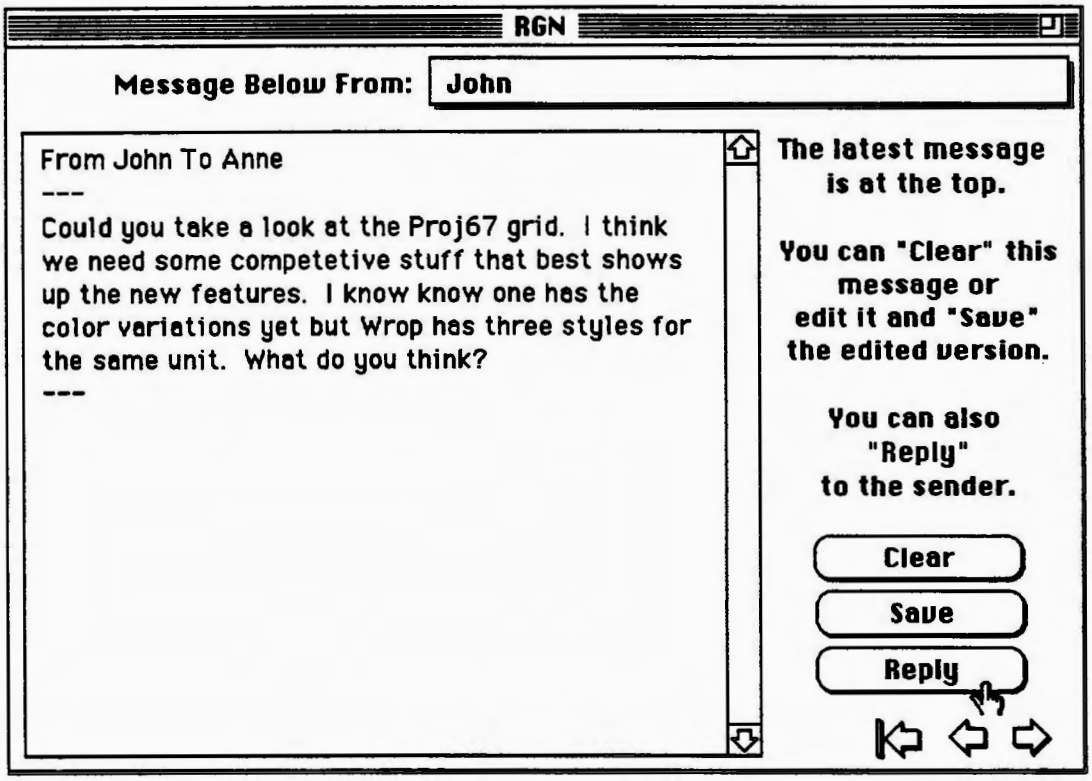


Fig.9 Incoming mail screen

Anne selects "Reply", goes to the screen shown in Figure 10, and types a reply.

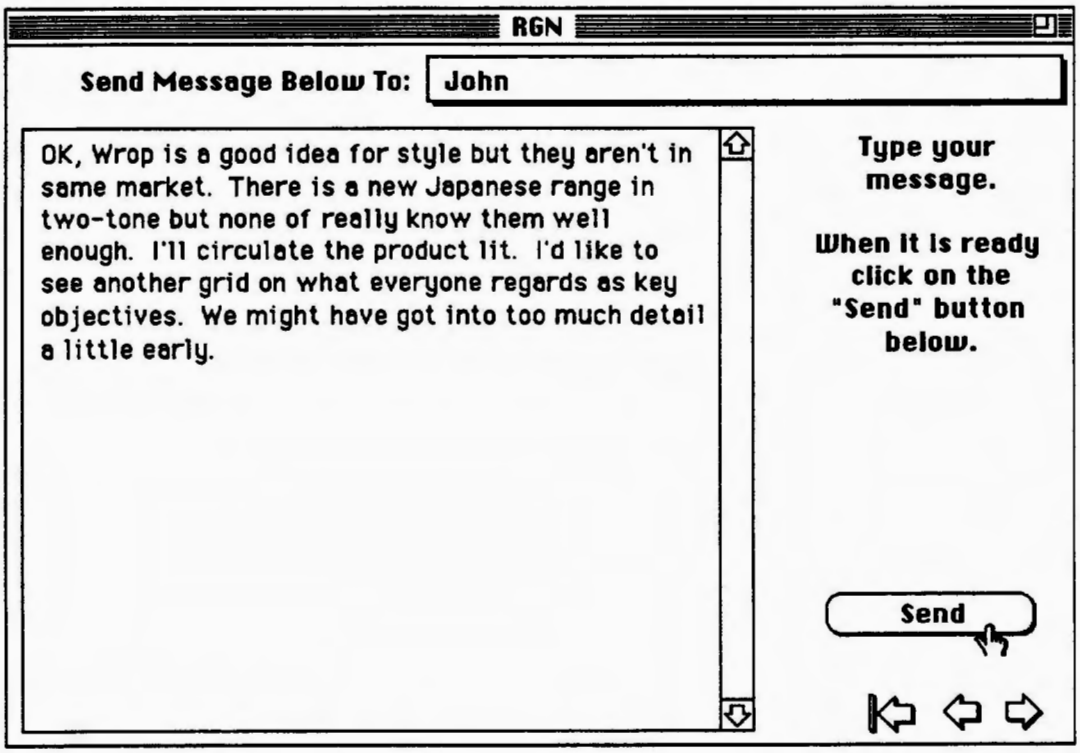


Fig.10 Outgoing mail screen

She selects "Send", returns to the screen shown in Figure 5, selects "SIGs", and goes to the screen shown in Figure 11.

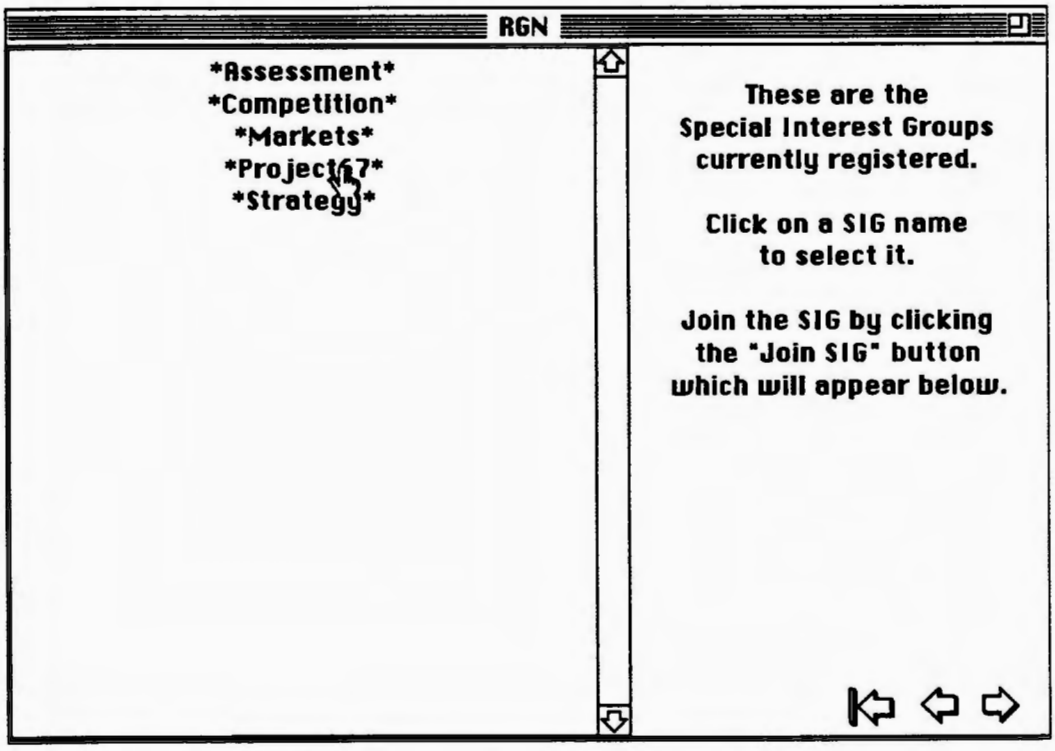


Fig.11 List of special interest groups screen

Anne clicks on "Project67" to access the specialist group concerned with product development and goes to the screen shown in Figure 12.

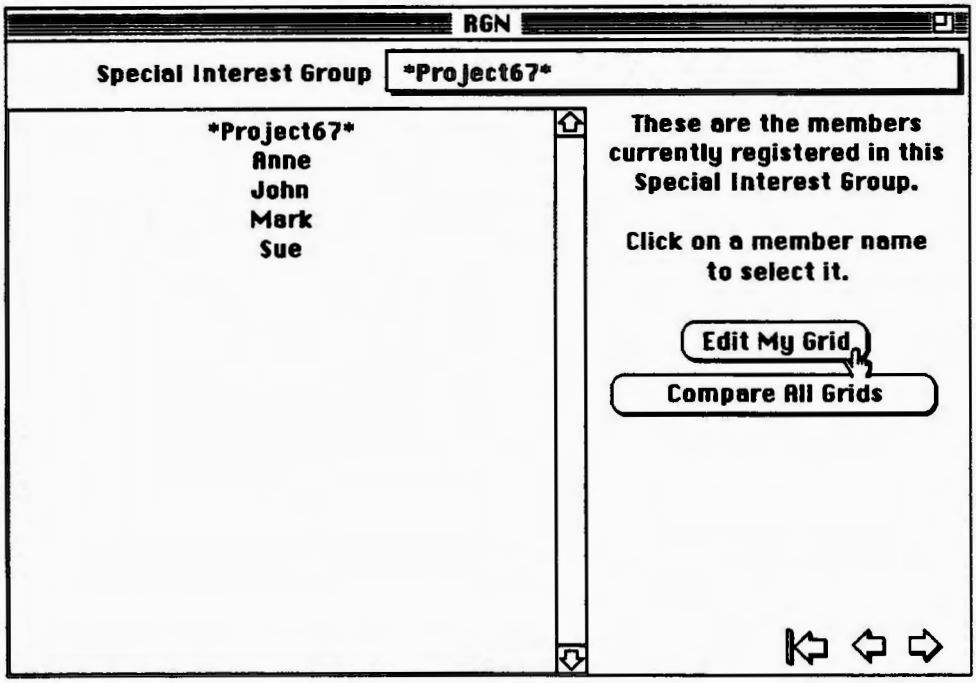


Fig.12 List of Project67 SIG members

She selects "Edit My Grid" and goes to the screen shown in Figure 13, accessing a grid she has already entered using the elements and constructs of the coordinator.

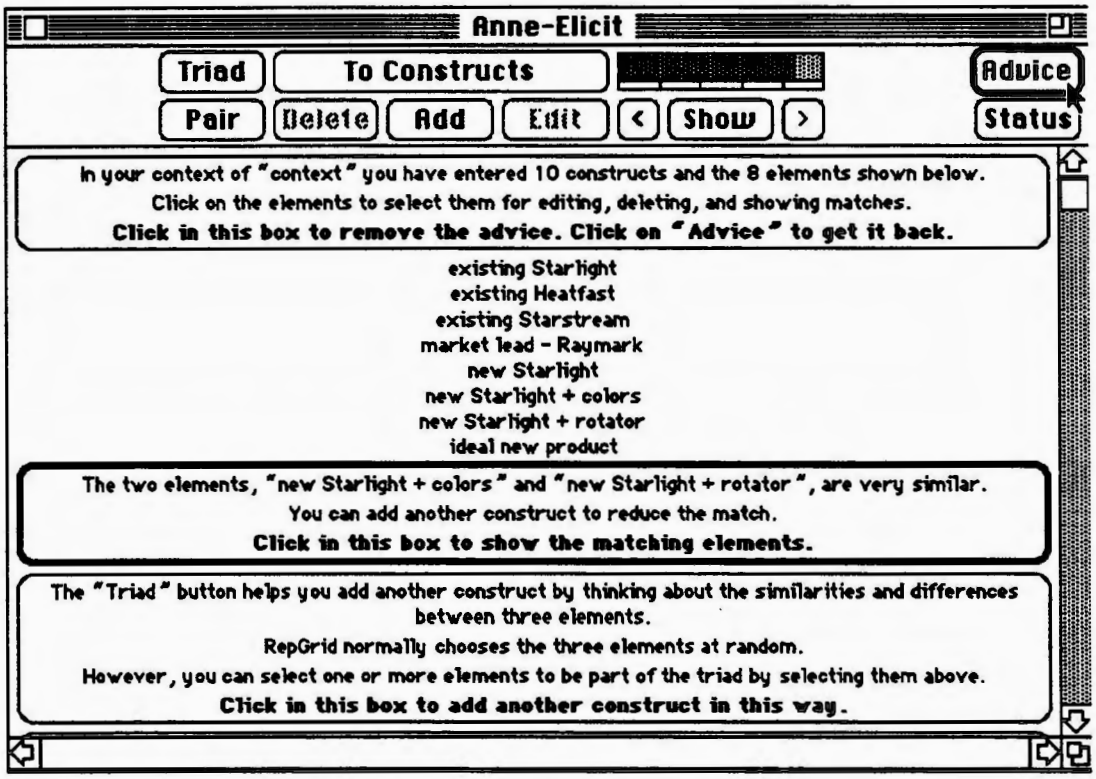


Fig.13 Repertory grid entities screen

Anne selects "Advice" and goes to the screen shown in Figure 14.

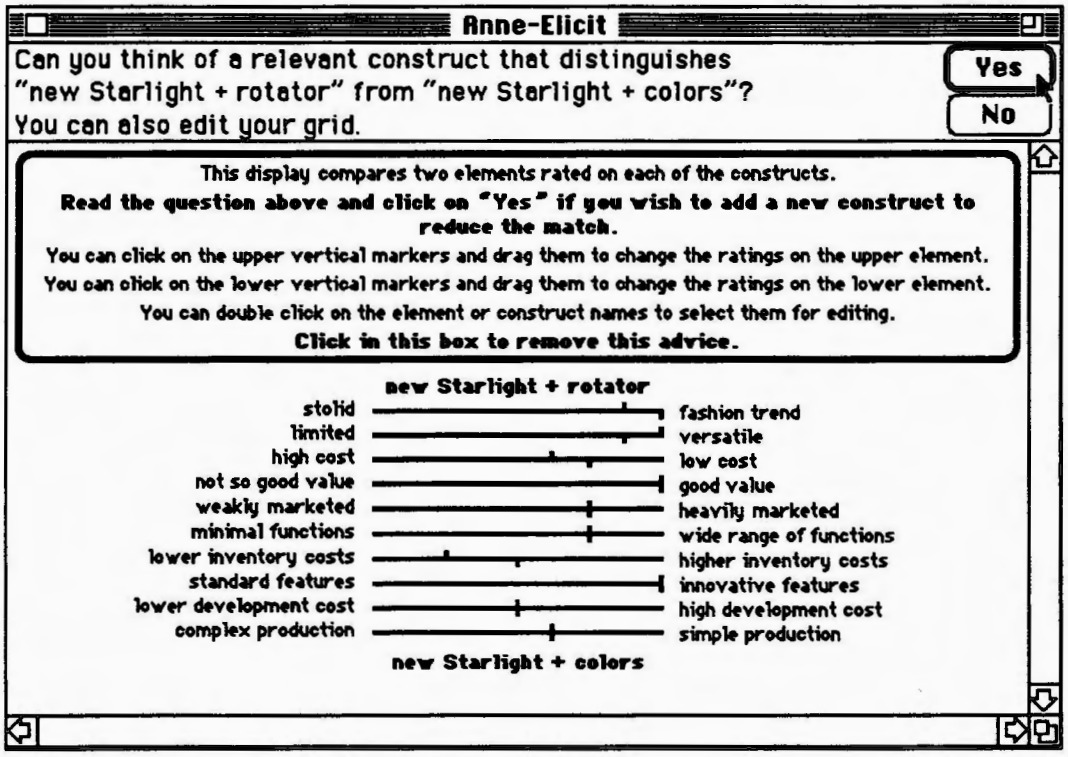


Fig.14 Element match screen

She selects "Yes" and goes to the screen shown in Figure 15.

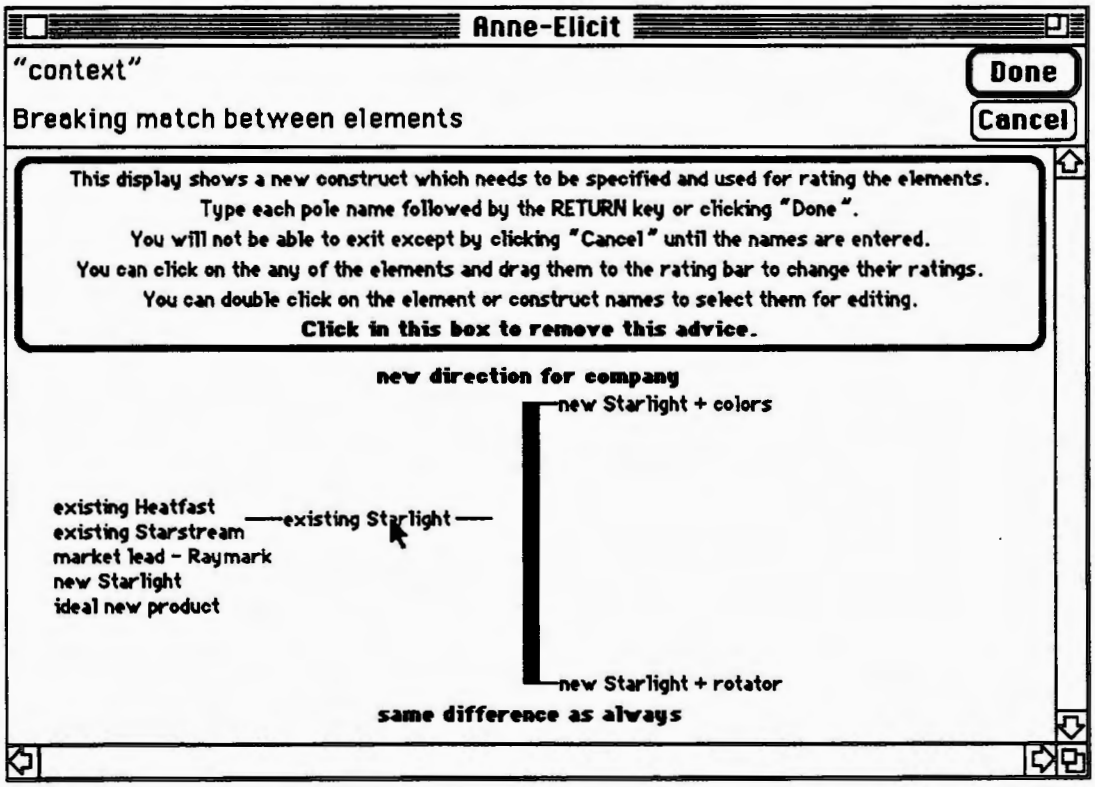


Fig.15 Element rating screen

When she has finished editing the grid, she returns to HyperCard as shown in Figure 16.

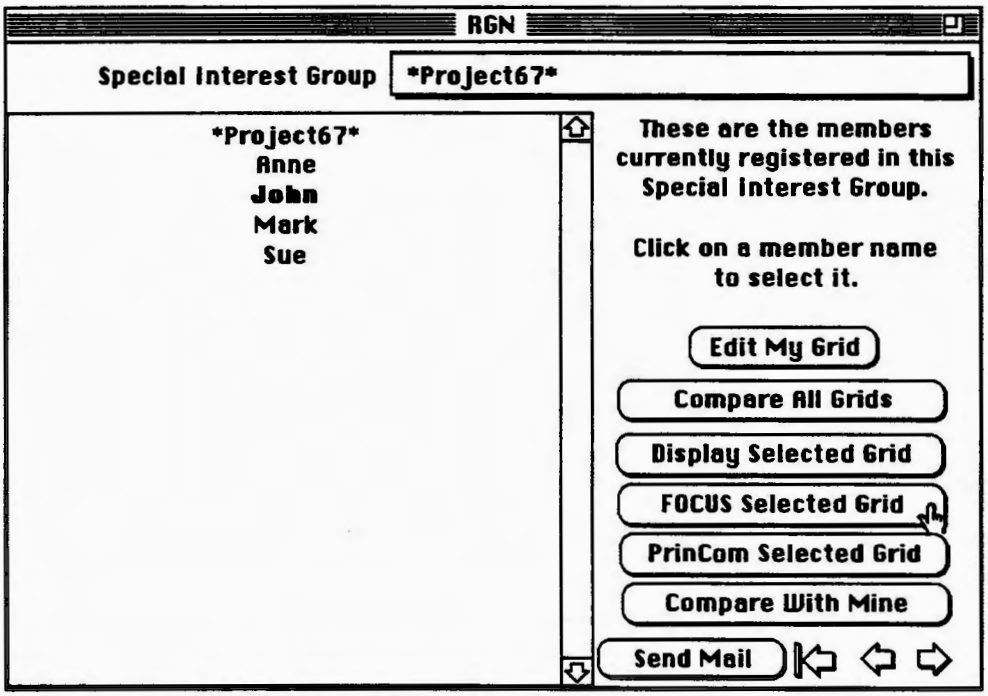


Fig.16 Grid analysis functionality screen

She clicks on "John", selects "FOCUS Selected Grid", and goes to the screen shown in Figure 17. This clusters John's grid and she can see that for him the three new models are all close and nearest to "ideal new product.'

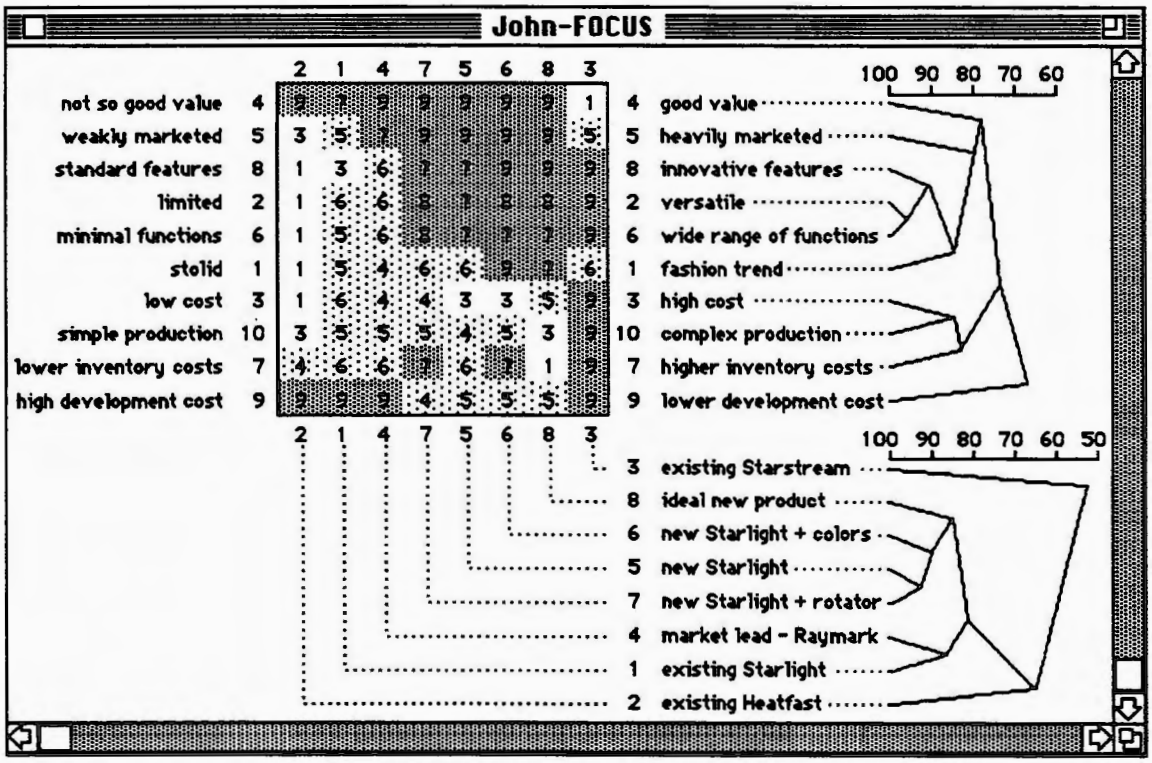


Fig.17 FOCUS cluster analysis screen

Anne returns to HyperCard, selects "Compare with Mine" and goes to the screen shown in Figure 18. It seems that she and John agree on most attributes except those involving costs at the bottom of the list, and on most products except the competition's market lead product. This comparison illustrates how the consensus-conflict axis in the left column of Figure 2 is determined and analyzed based on the participants' capabilities to use one another's terminology and distinctions in the same way.

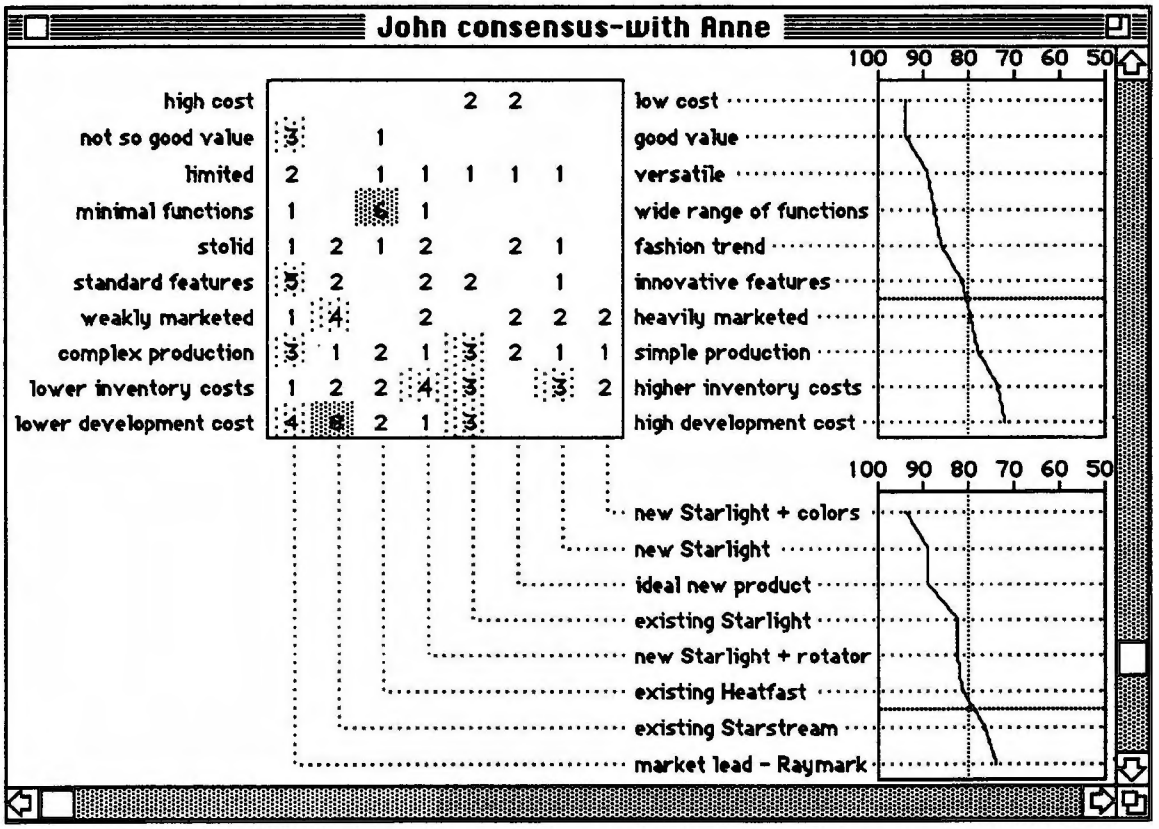


Fig.18 Grid comparison screen

She selects "Compare All Grids" and goes to the screen shown in Figure 19. An arrow from one person to another in the socionet shows that the first person can understand the constructs of the second. Thus, John is the only one with mutual understanding with everyone else. These comparisons are based on the correspondence-contrast axis of Figure 2. The capability to understand another person is assumed to depend on the availability of corresponding constructs that need not be consensual in that they may be termed differently. We have a basis for mutual understanding if we can make the same distinctions even if we have to determine a mapping between our terminologies to provide mutual translation.

Anne clicks on "Project67", the kernel grid provided by the chief executive's assistant, and uses a popup menu to return to HyperCard, and goes to the screen shown in Figure 16 with Project67 now selected. She can now explore why the Project67 grid is different from hers. She can send mail to the coordinator discussing the differences, querying his use of terms, and so on. The system is highly non-modal and the user can continue using the system as demonstrated, browse between cards, use the RepGrid tools to further develop grids, use the email to communicate, and so on.

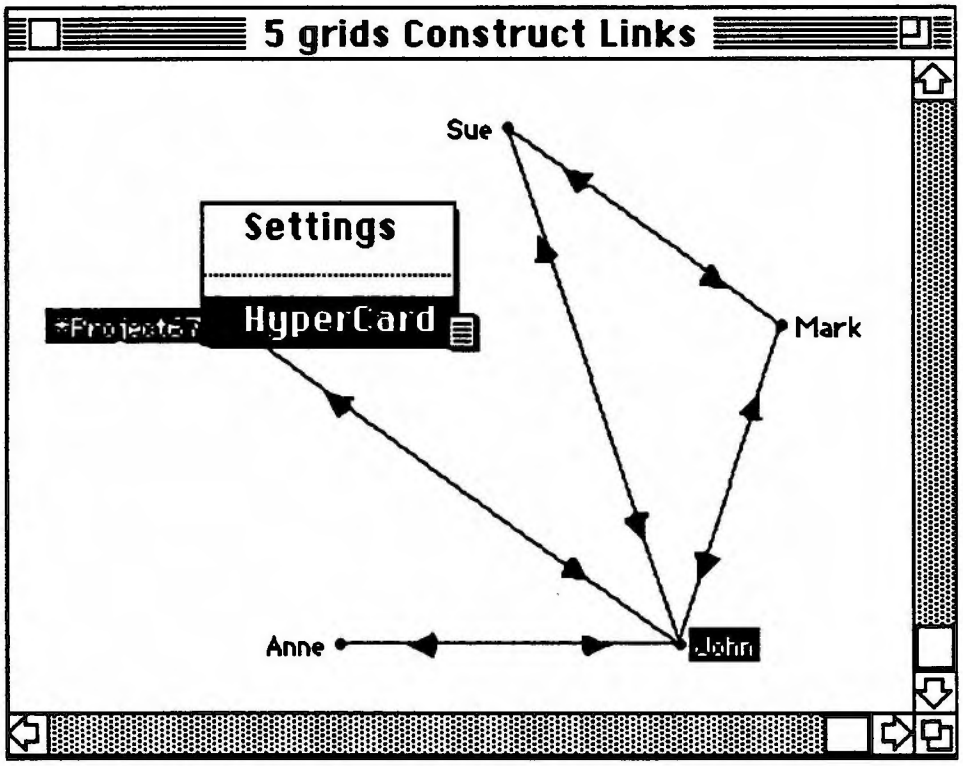


Fig.19 Socionet based on grid comparisons screen

CONCLUSIONS

Potential applications of knowledge acquisition systems already developed extend far beyond the development of expert systems; a significant area is the support of group processes through sharing and comparison of personal knowledge. RepGrid-Net is a message system supporting special interest group formation and collaborative activities at many levels. It integrates techniques and programs that have already been widely used to support the elicitation and analysis of conceptual structures. However, it makes them available in a very much more informal and open setting than previously.

The current implementation is written for the Apple Macintosh and operates only on a local server network. Transferring the code to other personal computers and workstations involves the usual problems of portability of highly interactive user interfaces. Remote operation presents no major problems since the grids are compact data structures similar in size to email messages, and work is already underway to allow the system to operate over global networks.

The system described shows the potential current applications of computer-mediated communication systems to be extended simply and naturally with a range of new features supporting human knowledge processes.

ACKNOWLEDGEMENTS

Financial assistance for this work has been made available by the Natural Sciences and Engineering Research Council of Canada. We are grateful to a number of colleagues who have used RepGrid-Net and provided us with data and critical feedback. In particular, Saul Greenberg has provided a number of significant insights.

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. (1989). A survey of knowledge acquisition techniques and tools. **Knowledge Acquisition 1** (1), 39-58 (March).

Boose, J.H. & Bradshaw, J.M. (1987). Expertise transfer and complex problems: Using AQUINAS as a knowledge acquisition workbench for knowledge-based systems. **International Journal of Man-Machine Studies, 26,** 328.

Boose, J.H. & Gaines, B.R., Eds. (1988). **Knowledge Acquisition Tools for Expert Systems**. London, Academic Press.

Boose, J.H. & Gaines, B.R., Eds. (1990). **The Foundations of Knowledge Acquisition**. London, Academic Press.

Brown, G.S. (1969). **Laws of Form**. London: George Allen & Unwin.

Gaines, B.R. (1988). Knowledge acquisition systems for rapid prototyping of expert systems. **INFOR, 26**(4), 256-285 (November).

Gaines, B.R. (1989) Social and cognitive processes in knowledge acquisition. **Knowledge Acquisition 1**(1), 251-280 (March).

Gaines, B.R. (1991) Between neuron, culture and logic: explicating the cognitive nexus. **Proceedings of ICO'91: Cognitive Science: Tools for the Development of Organizations**. pp. 222-32. Montreal: GIRICO.

Gaines, B.R. & Boose, J.H., Eds. (1988). **Knowledge Acquisition for Knowledge-Based Systems**. London: Academic Press.

Gaines, B.R. & Linster, M. (1990). Development of second generation knowledge acquisition systems. Wielinga, B., Boose, J.H. & Gaines, B.R., Schreiber, G., van Someren, M., Eds. **Current Trends in Knowledge Acquisition.** pp. 143-160. Amsterdam, IOS.

Gaines, B.R. & 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. (1989). Comparing the conceptual systems of experts. **Proceedings of the Eleventh International Joint Conference on Artificial Intelligence**. pp.633-638. San Mateo, California: Morgan Kaufmann (August).

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

Heller, A (1990). **Can Modernity Survive?** Berkeley: University of California Press.

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

Kerr, E.B. & Hiltz, S.R. (1982). **Computer-Mediated Communication Systems**. New York: Academic Press.

Liou, Y.I., Weber, E.S. & Nunamaker, J.F. (1990). A methodology for knowledge acquisition in a group decision support environment. **Knowledge Acquisition 2** (2), 129-144 (June).

Pask, G. (1975). **Conversation, Cognition and Learning**. Amsterdam: Elsevier.

Popper, K.R. (1974). Autobiography of Karl Popper. Schilpp, P.A., Ed. **The Philosophy of Karl Popper**. pp. 3-181. La Salle, Illinois, USA: Open Court.

Popper, K.R. (1974). Autobiography of Karl Popper. Schilpp, P.A., Ed. **The Philosophy of Karl Popper**. pp. 3-181. La Salle, Illinois, USA: Open Court.

Quartermain, J. S. (1990). **The Matrix: Computer Networks and Conferencing Systems Worldwide**. Digital Press.

Shaw, M.L.G. (1980). **On Becoming a Personal Scientist**. London: Academic Press.

Shaw, M.L.G. (1985). Communities of knowledge. Epting, F. & Landfield, A. **Anticipating Personal Construct Psychology**. pp.25-35. Lincoln: University of Nebraska 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. (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. (1989). A methodology for recognizing conflict, correspondence, consensus and contrast in a knowledge acquisition system. **Knowledge Acquisition 1**(4), 341-363 (December).

Simmel, G. (1950). **The Sociology of Georg Simmel**. (trans. Wolff, K.H.) New York: Free Press.

Wolff, K.H. (1976). **Surrender and Catch: Experience and Enquiry Today**. Holland: D.Reidel.

USING SCRIPT-BASED OBJECTS FOR KNOWLEDGE DESCRIPTION IN GSE

Glenn G. Shephard, College of Business
San Jose State University, San Jose, CA 95192-0069

ABSTRACT

Recent exploratory research developed and tested a guided self-elicitation (GSE) methodology. With GSE, an expert is enabled to capture his/her own performed expertise as production rule-instances. Experimentally self-elicited decision analyst expertise (leading subjective probability assessment interviews) demonstrated that complex performed expertise can be described in rule-processed form. An existing limitation on GSE's ability to enable expert reporting of knowledge content and structure is addressed. A script-based object model extension to GSE for describing complex self-elicited expert knowledge is proposed and discussed.

INTRODUCTION

GSE seeks to enable a performing expert to self-elicit his/her own performed expertise through a process of "guided self-elicitation." The knowledge engineer's role in this process is to transfer cognitive processing and self-elicitation process knowledge to the expert for the expert's own use in reporting his/her own task directing cognitive processing.

Published research findings underlie the GSE methodology. Experimentation demonstrated the viability of GSE in capturing the guiding expertise of a consulting decision analyst as performed while leading a business person to probabilistically encode an uncertain business variable during a subjective probability assessment interview. The actual performed expertise of an experienced decision analyst, as practiced in three assessment interviews, was captured as production rule-instance traces.

The self-elicited guidance expertise for probability assessment interviewing has shown itself to be complex and multifaceted. The experienced decision analyst seems to reason quantitatively with probability requirements, qualitatively to compensate for the biasing affects of cognitive heuristics and heuristically with experienced-based interview pacing; all in orchestrated combination.

Initial GSE guidance protocols proved incomplete and hampered the decision analyst in describing the structure and meaning of his internal knowledge processing. A script-based object model has been developed to enable robust recording of arbitrarily complex forms of knowledge and cognitive processing that may be used by experts in rule-based processing. Two revised GSE generalization exercises have been developed that facilitate self-elicitation description of complex knowledge.

GSE METHODOLOGY

Verbal reporting of prior cognitive processing is untrustworthy [Nisbett and Wilson, 1977][Ericsson and Simon, 1980,1984]; making knowledge "engineered" from such sources suspect [Vandamme, 1987]. This problem is obviated by GSE. Guided self-elicitation (GSE) uses retrospective prompting of an expert with his/her own recorded prior task perceptions to rehearse ("rote rehearsal" per [Zechmeister and Nyberg, 1982]) and reconstitute the stream of prior cognitive information processing for authentic expert reporting. This works because people are able to concurrently report their actual cognitive information processing but cannot do so retrospectively or introspectively [Nisbett and Wilson, 1977][Ericsson and Simon, 1980,1984].

GSE uses videotape recording and editing technology to enable an expert to self-elicit his/her own performed expertise. Videotaped visual and aural task perceptions provide the basis for creating and labeling "expertise review tapes" (ERTs) used by the expert for rehearsing and reporting his/her recorded performances. GSE requires the expert to perform a series of focused exercises to identify, label, interpret and associate his/her recorded perceptions into performance traces of rule-instances.

Published cognitive psychological research underlies the knowledge elicitation transfer protocols used in GSE. Figure 1 shows an adaptation of the Short Model [Gilmartin, Newell and Simon, 1976] that forms the basis of the qualitative model of conscious, rule-based cognitive information processing used to guide expert self-elicitation. It is an easily understood framework, a comfortable metaphor for the self-eliciting expert.

In the qualitative processing model, meaningful information structures are identified in the flow of visual and aural perception during task performance as categorized entities [Bruner, Goodnow and Austin, 1956]. Entities are treated by the model as discrete information units that are subject to cognitive rule-based processing. GSE calls these entities knowledge elements (KELs)[1].

---

1 KEL has been adopted for abbreviating the GSE "knowledge element" instead of KE to avoid confusion with common usage of KE to abbreviate "knowledge engineer."

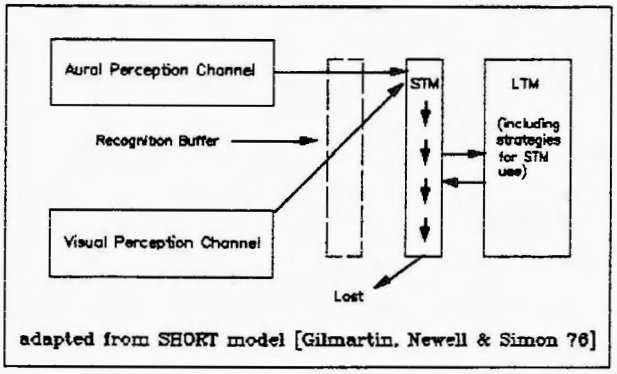


Figure 1. Conscious, Rule-Based Cognitive Information Processing

There can be both external, perceptual KELs and internal, non-perceptual KELs. Whichever their source, KELs enter a finite information processing system (IPS) short term memory (STM) queue as symbols - which pass through and are lost [Newell and Simon, 1972,1976]. Perceptual KELs enter the IPS model as stimulus pattern signals through the visual and aural perception channels. They pass through the recognition buffer where they are assigned symbol identities which enter the STM queue. Internal KELs are generated through the internal thought processes of the expert and do not involve perceptual inputs. But they do contribute KELs as symbols to the STM queue.

The GSE-IPS model has limited STM queue capacity [Miller, 1956]. The limit is "seven plus or minus two" compiled information "chunks" at any time. As Miller "chunks" correspond closely to GSE KELs, this constraint provides a useful gauge for judging KEL granularity in self-elicited expertise that is reported as rule-instance performance traces.

GSE methodology requires the participating expert to iterate through a series of focused ERT review exercises. Each GSE exercise requires a different review focus where the expert is directed to attend specifically to one aspect/stream of his/her rehearsed thought processes - excluding all others. Such single-minded focusing of expert attention to one specific part of his/her ongoing processing enables the expert to verbally report on this aspect of reasoning [Anderson, 1990][Ericsson and Simon 1980,1984].

DEVELOPMENT BACKGROUND

GSE was developed in experimental research that recorded and analyzed the performed natural expertise of an experienced decision analyst acting as a consultant while leading a series of subjective probability assessment interviews [Shephard, 1990,1991]. In each, the analyst helped a different business expert (a "client") overcome cognitive biases that are inherent in conventional thinking about subjective feelings toward uncertainty [Tversky and Kahneman, 1974].

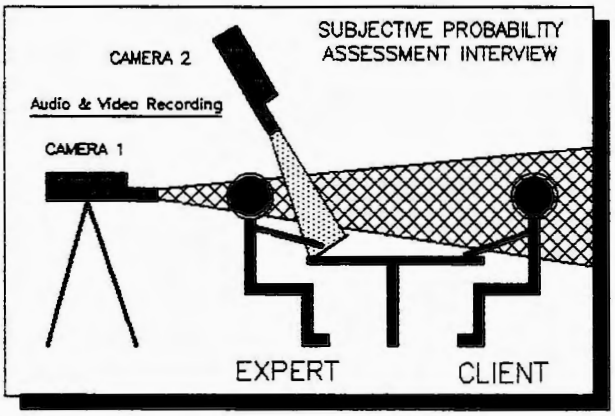
Three subjective probability assessment interviews with three different production planners who worked in the same department of a large Aerospace Company were subjected to GSE. One uncertain variable was assessed in each interview; a different, but similar variable for each. The procedure used by the decision analyst was a slightly modified version of published SRI/SDG protocols for subjective probability assessment [Merkhofer, 1987][Stael von Holstein and Matheson, 1978][Spetzler and Stael von Holstein, 1975]. The decision analyst led the assessor clients by (1) procedurally directing the encoding process and (2) assisting the clients in compensating for their displayed affects of cognitive heuristics normally employed during the encoding process.

The GSE methodology consists of three exercises: identification-labeling, association and generalization. Tailored for an interviewing process, GSE was storyboarded to assist explanation of videotape recording and the GSE exercises. All three interviews were set up and run as shown in Storyboard 1. Sets of videotape recordings of decision analyst visual and aural perceptions during the interviews were used to prepare ERTs for expert GSE exercises. See Storyboard 2.

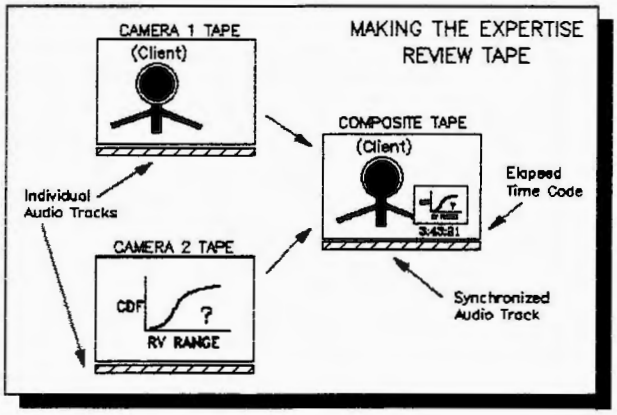
Identification-Labeling Exercise

The first step in GSE is identification-labeling. The analyst transduced and translated his videotaped interview perception patterns (visual and aural) into videotape frame sequence blocks of labeled external cues and personal interventions (actions). See Storyboard 3.

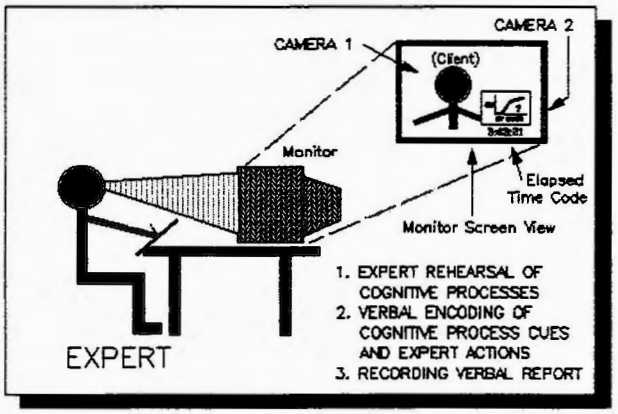
The decision analyst recorded his identification-labels on a prepared worksheet. It provided space for recording current operating goal, start time of next cognitive processing element and short descriptions for external perception cues and analyst interventions.



Storyboard 1



Storyboard 2



Storyboard 3

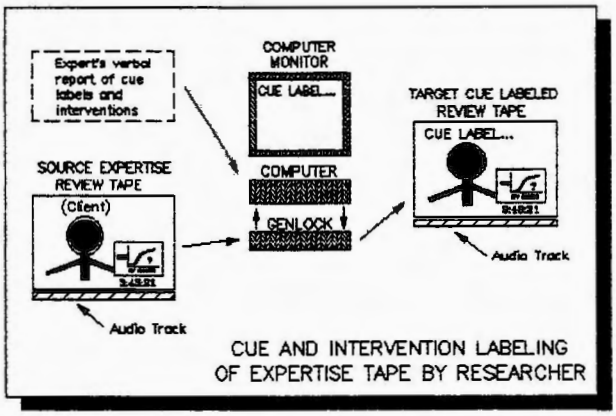
From dialogue transcribed from the ERTs and the analyst's worksheets, formal written protocols of the interview performances were created. The dialogue of each interview was used as a proxy for the decision analyst's full videotape recorded visual and aural perceptions. The analyst's labels were also overlaid onto the ERTs as shown in Storyboard 4. The resulting screen display is shown in Figure 2. The new recorded copies are called "labeled expertise review tapes" (LERTs).

After the LERTs were created, the analyst viewed them again and recognized that another level of knowledge processing was operating during the interviews; concurrent with that already reported. The identification-labeling exercise was repeated by the decision analyst.

The second identification-labeling exercise produced additional labeled KELs relating to the decision analyst's "pacing" of the assessment interviews. These were added to the existing expertise performance protocols and LERTs.

Association Exercise

The second step in GSE is the KEL association exercise. It required the decision analyst to rehearse his "fully labeled expertise review tapes" (FLERTs) and to record his associations on the fully labeled expertise performance protocols.



Storyboard 4

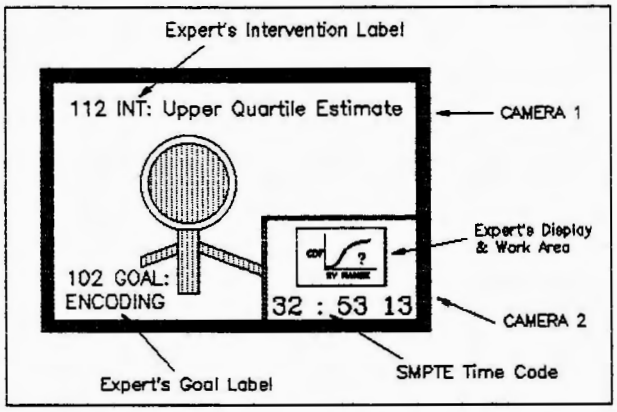


Figure 2. Labeled Expertise Review Tape Screen

During the association exercise, the analyst identified groups of prior KELs that were causally associated and responsible for each rule-instance's execution and observed intervention. Figure 3 shows an example of one of the decision analyst's KEL associations.

Decision analyst expertise: Using GSE, the decision analyst described his own performed expertise as rule-instance traces. Two kinds of KEL were identified: external "perceptual" KELs and internal "non-perceptual" KELs. Internal KELs were recognized and reported by the analyst only after he had "peeled-back" the layer of external perceptual KELs and focused his attention on the cognitive processing activity that remained. The analyst described this level of internal cognitive processing as management of interview pacing; a process in which an initial "store" of client "energy" (interest, willingness

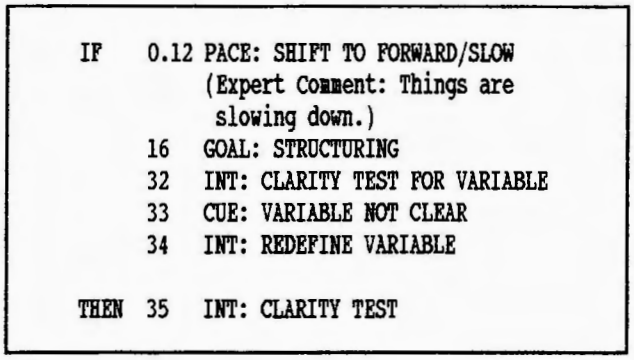


Figure 3. Decision Analyst's Formulated Rule-Instance Association

to participate, patience, etc.) is carefully spent by the analyst during the interview to achieve effective client probability assessment performance. The consequence of pace processing was the ongoing revision of the interview agenda operation in terms of agenda processing direction and speed.

The process of interview pacing generated symbols that joined with perceptual KELs in a single cognitive processing queue. The analyst associated both external and internal KELs into the same rule conditions when composing the rule-instance descriptions that explained his interview performances (see Figure 3). The two concurrent thought processes combined to direct his overall rule-driven task performance behavior.

The reasonableness of the decision analyst's self-elicited rule-instance trace descriptions was explored with a simple type of granularity test. Drawing a correspondence between GSE KELs and Miller's information chunks [Miller, 1956], it was expected that the number of KELs per rule instance should satisfy a "seven plus or minus two" STM queue size constraint. This was found in all but one of the 238 rule-instance associations generated by the decision analyst.

Generalization Exercise

The generalization exercise is the third step in the GSE methodology. It was developed to enable an expert to decompile and report the structured knowledge of his/her skeletal KELs (see example skeletal KEL in Figure 3). Initially the decision analyst was introduced to attribute-value (A-V) and object-attribute-value (O-A-V) knowledge representation forms [Harmon & King, 1985]. They were proposed as descriptive vocabulary for his use. He was asked to review his FLERTs and focus his attention on any and all interview performance attributes or factors that came to mind and seemed relevant to his thinking during the interview process.

From his first review of a single interview's FLERT, the analyst generated a preliminary list of 20 task performance related factors that seemed generally pertinent (see Figure 4). He indicated that concern for the various performance factors, singly and in groups, surfaced in his mind at different times during the interview; first blossoming and then receding from conscious attention. When asked, the analyst found difficulty in locating the factors in specific KELs and was unable to structure the factors into meaningful KEL explanations. In short, the A-V and O-A-V descriptive language lacked necessary descriptive power.

1. Attitude
2. Experience
3. Brightness
4. Understanding of probability
5. Memory
6. Depth of procedural thinking about job that he does
7. Introspectiveness (intuitive vs analytical)
8. Independence of thinking (independent vs job driven)
9. Ability to adjust thinking from heuristics (can they see the illusion)
10. Prior preparation for elicitation process
11. Breadth of view of job (tunnel vision on tasks)
12. Degree of interest in process of elicitation (curiosity)
13. Attention to detail (picky vs sloppy)
14. Degree of self-consciousness (ability to role-play)
15. Complexity of idea being presented (simple to complex)
16. Candor
17. Clarity of expression
18. Length
19. Ability to speculate (creative/free association thinking)
20. Degree of connectedness with other intercepts (stand-alone vs part of single idea)

Figure 4. Preliminary Task Performance Factor List

An attempt was made to expand the A-V/O-A-V language by discussing the concepts of feature analysis [Anderson, 1990] and by suggesting a repertory grid-like approach for KEL attribute elicitation [Hart, 1986]. Based on this expanded guidance, the analyst again reviewed the same interview FLERT and refined his preliminary list into a factor dictionary with ten factor entries. (See Figure 5 for an excerpt from the decision analyst's factor dictionary.) In this case, each factor entry includes a unique value range definition. The expert still found it very difficult to structure meaningful KEL descriptions or to confidently map factors into specific KELs. The A-V/O-A-V language was obviously inadequate for its purpose. A more robust language was clearly needed.

Experience/familiarity with behavior of product
1) Significant experience but not recent -> fuzzy recall
2) Limited experience but recent
3) Significant recent experience
4) Significant recent experience and some personal analysis of the experience

Mental acuity - general
1) Average
2) Above average
3) Quick

Understanding of uncertainty/probability at intuitive level
1) No grasp of probability as a representation of uncertainty or variability
2) Limited grasp
3) Intuitive appreciation of function of probability

Perspective of job responsibilities and activities
1) Completely job-driven; has perspective of immediate manager or "job description" book; "company man"
2) Mixed
3) Independent/personally objective view of job responsibilities

Figure 5. Decision Analyst's Factor Dictionary

KEL DESCRIPTION AND REPRESENTATION

Pre-generalization GSE exercises capture performed expertise as skeletal rule-instance traces. Individual rule-instances are described by collected KEL labels that are associated into condition-action rule formats of varying complexity (as measured by the number of condition elements per rule instance) [Shephard, 1990]. Both knowledge content and structure of reported KELs stand unrevealed at this point.

Prior research reports that cognitive knowledge structures increase both in size and effectiveness as expertise develops [de Groot, 1978] [Chase & Simon, 1973] [Newell & Simon, 1972] [Simon & Gilmartin, 1973]; ie, as they are compiled. Further, [Neches, Langley & Klahr, 1987] conclude that "in principle, the condition sides of production rules [can] be arbitrarily complex..." Based on this, GSE proposes that KELs may contain the arbitrary complexity of expert knowledge and yet process straight-forwardly through a single channel Miller-constrained STM queue [Miller, 1956]. In support of this approach, GSE experimentation has demonstrated successful expert use of single channel rule processing (see Figure 1) to describe

reasonably complex knowledge processing. If rule processing with KELs is to be simple and straight-forward, then complexity of description must be expected for KELs and a suitably robust descriptive language must be provided.

The object is proposed as a plausible, robust candidate for describing individual GSE KELs. An object is "...something you can do things to..." [Booch, 1991] (such as a KEL) "...with crisp boundaries and meanings for the problem at hand..." [Rumbaugh, Blaha, Premerlani, Eddy & Lorensen, 1991]. This is a good definition of KELs created during GSE identification-labeling of videotape frame sequences (see GSE identification-labeling exercises discussed earlier) and used in expert rule-based cognitive processing.

Reflecting the nature of human cognition, the GSE KEL description exercise must cause the expert to focus his/her attention in turn on the external, cue-designated and then the internal, intervention-designated dimensions of his/her own cognitive processing. Phenomenologically this corresponds closely to the perceptual (external) dimension and the combined knowledge-based/memory/motoric (internal) processing dimensions reported by [Ericsson & Smith, 1991]. This compressed phenomenological structure is then used as the organizing basis of the KEL object model to be used in subsequent knowledge acquisition recording.

Two exercises have been developed based on new pilot test experimentation that should enable an expert to self-elicit the content and structure of his/her own KELs. The pilot test experts are the pilot test and experimental experts who originally participated in GSE development and who used their original LERTs, FLERTs and written protocols in this latest testing. The pilot test results have served as the basis for preparing two exercise protocols for planned future experimentation.

KEL Description

The object model is proposed as a comprehensive recording knowledge representation form for GSE KELs; a form that can encapsulate the abstracted essences of an entire complex module of perceived reality. As such, it can be used to record public and private data structures of all types as well as public and private methods that may describe both cognitive and behavioral procedural knowledge. For GSE KELs, the object model is realized in the form of a recording template for both self-elicited external (perceptual) and internal (knowledge-based/memory/motoric) knowledge.

Reportable cognitive processes occur through time. The information content and knowledge structures that the expert recognizes, identifies and processes during the internal GSE KEL description exercise are experienced as sequences of events. When their ordering, their sequencing in time is relevant, they must be

reportable in the time sequence in which they occur. For recording, this suggests using an underlying, base structure within the object representation such as Schank and Abelsons' scripts [Schank & Abelson, 1977]. Scripts in the expert's use can provide the coordinating function needed to organize the relationships of otherwise seemingly disparate groups of variously organized knowledge and information as well as the methods and messages that must have operated on them.

In general, scripts are knowledge structures that describe sequences of events within set contexts [Schank & Abelson, 1977]. The typical elements of a script include "...entry conditions, props, roles, tracks, and scenes..." [Turban, 1992]. Props and roles describe the background context of scripted situations, tracks deal with script options that exist in master scripts and scenes describe actual events. In GSE KEL description exercises, the knowledge engineer will record the expert's reported KEL content and structure as scripted sequences of events (micro-scripts) in which other knowledge forms combine in scenes that describe the various types of knowledge processing and reasoning described above by Slatter and Mayer [Slatter, 1990] [Mayer, 1992].

In the two dimensional GSE object template discussed above, the perceptual dimension section will be used primarily to describe the source of external KEL script entry conditions and ongoing data inputs and outputs. The internal knowledge-based/memory/motoric section will be used to describe scene-by-scene cognitive processing episodes that occur (during expert task performance) sequentially throughout each KEL script. Processing in this dimension can also generate perceptually-based script entry conditions as well as produce internal, non-perceptual entry conditions for future processing.

Describing External Perception

Categorization is the process that provides humans with effective management of their powerful capabilities for stimulus discrimination and provides the basis for purposeful human behavior [Bruner, Goodnow & Austin, 1956]. It is the process of abstracting meaningful information/knowledge structures from perceptual reality. It is an aspect of the mental process called "chunking" which Miller describes as "...organizing... stimulus input simultaneously into several dimensions and successively into a sequence of chunks..." [Miller, 1956].

The perceptual dimension of the human cognitive processing system is where visual and aural input is received, identified, categorized and recognized as surface feature components of particular perceptual (external) KELs or is implicitly mapped to internal performance factors models. The perceptual section of the GSE KEL object template is used to record an expert's interpreted meaning that is extracted from the ongoing, time-based flow of external,

environmental stimuli he/she receives. It occurs in the recognition buffer shown in Figure 1. The process itself is one that classifies and categorizes time-streamed perceptual stimuli as recognizable information structures which serve as input to the expert's ongoing cognitive processing. These structures serve as entry conditions for script activation.

Categorization as Communication: The initial conception of the KEL object as a single, powerful representation form for expert description of self-elicited task performance [Shephard, in press], did not provide an "expert-friendly" metaphor for guiding expert self-elicitation performance. A new approach was needed. A communication metaphor was selected as a focusing device for feature identification and reporting within the task context.

For GSE, we will consider the recognition/categorization process as one of communication in which an expert acts as a receiver for his/her perceptual environment. He/she continuously scans and decodes <u>all</u> time-streamed, perceptual stimulus pattern input into meaningful messages. Appropriate encoding/decoding process models for such communication are discussed in [Haney, 1992] and [Malandro, Barker & Barker, 1989]. Figure 6 shows an adaptation of the communication process model for the GSE recognition process.

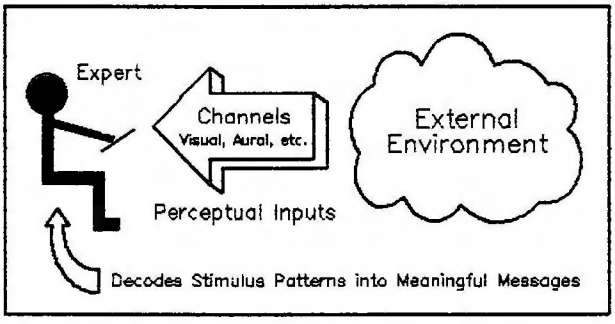


Figure 6. GSE Communication/Recognition Process

Viewed as communication, visual and aural stimulus inputs divide into the verbal (symbolic) and non-verbal (non-symbolic) forms [Malandro, Barker & Barker, 1989]. In turn, these categories can be subdivided into vocal and non-vocal forms of communication. Malandro, et. Al. describe the resulting combinations as a set of four possible communication behaviors:

> "1. verbal/vocal: include the actual words or symbols that are used when communicating with another,
>
> 2. verbal/non-vocal: [syntactically informal] but ha[s] much in common with verbal/vocal,
>
> 3. non-verbal/vocal: include all cues in oral and written speech other than the actual content itself and

4. non-verbal/non-vocal: includes all of those behaviors which are used to communicate a message--other than through the voice."

Verbal communication is relatively straightforward but is responsible for 30 percent or less of all messages we receive when spoken to [Steele, Murphy & Russill, 1989]. The bulk of the messages we receive are conveyed through non-verbal body language, including gestures, facial expressions, posture, etc. The categories of non-verbal communication behavior that may be specifically relevant to the subjective probability assessment interview domain, for example, are listed below as sets of incoming message/features/cues:

1. environment
2. body type, shape and size
3. clothing and personal artifacts
4. body movement and gestures
5. facial expression and eye behavior
6. voice characteristics and qualities

Incoming message/features/cues are generally combined into clusters for reliable meaning. Isolated, single cues can be misinterpreted and the signals they send incongruous. When combined, their apparent meaning is more reliable [Steele, Murphy & Russill, 1989]. Taking non-verbal gestures and other body signals as an example, Steele, et. al. suggest the following message cluster as a fairly accurate indicator of *cooperation/willingness/relaxation*:

* "Head is inclined to one side
* Body leans forward in chair
* Jacket or coat is unbuttoned
* Palms of hands are opened and upturned
* Sits at right angles to or directly alongside other person."

Describing Internal Knowledge-based/Memory/Motoric Processing and Behavior

The internal GSE object template section provides for description of both the cognitive processing activity and motor behavior reported by a self-eliciting expert during the internal generalization exercise. The description process (performed by the knowledge engineer in conjunction with the reporting expert) involves scripting the expert's reported cognitive processing and behavior into a sequence of separate knowledge-based/memory/motoric events or scenes and describing the various types of knowledge involved in each. Surface, standard boiler-plate type procedural knowledge, "deep" procedural knowledge including quantitative and/or qualitative processing, mental modeling, etc. may be described in scripted scenes. Also described are the results of script operation and possible sub-scripting alternatives.

A single, comprehensive script-based object template is proposed for reporting all KELs (see Figure 7). The self-eliciting expert reviews his/her FLERTs with their corresponding written protocols while performing the external (for cues) and internal (for interventions) generalization exercises. The reported results of the exercises are then jointly organized by the knowledge engineer and expert into KEL knowledge descriptions using the object recording template.

Generalization Exercises

Based on the recently completed pilot tests, two separate generalization exercises have been developed. The first exercise asks the expert to focus on external perceptual KELs labeled as cues on LERTs and FLERTs. The second exercise asks the expert to focus his/her attention on internal knowledge-based/memory/motoric KELs labeled as interventions on LERTs and FLERTs. No conclusion has been reached whether one exercise should be performed before or after the other.

The two pilot experts were unable to use the object template shown in Figure 7 as a meaningful checklist for self-elicitation guidance. It was too complex. Based on their experience, a simplified checklist was developed for each exercise that provide the experts with useful structure for reporting KEL content and structure:

Internal Generalization Checklist:

What is happening
How is it happening
Why is it happening
Critique of process

External Generalization Checklist (This list has not been tested by the pilot experts, but is suggested by one of the expert's successful use of the internal checklist.):

Message description
Vocal cues (non-symbolic)
Non-verbal symbolic cues
Non-verbal non-symbolic cues

THE NEXT STEP

Planning is currently underway to retest the entire GSE methodology that has been developed by re-performing a new series of experimental subjective probability assessment interviews using new subjects and decision analyst experts. If this proves impractical, a new domain will be selected and a new experiment designed.

KEL Identification:

1. Label description of KEL.
2. Explanation of meaning & significance of KEL.

Perceptual Dimension Section: (ANALYZED AS VERBAL & NON-VERBAL MESSAGES TO GENERATE SCRIPT ENTRY CONDITIONS)

3. Identification & description of videotape recorded perceptual message clusters (visual & aural) recognized as verbal and non-verbal communication.
4. Identification & description of individual features comprising each identified message cluster.

Knowledge-Based/Memory/Motoric Dimension:

Knowledge-Based/Memory Section: (SCRIPT-BASED DESCRIPTION WITH PROPS, ROLES, TRACKS, SCENES & ENTRY CONDITION REQUIREMENTS) (PRIVATE SCRIPTS/SCENES)

5. Discuss KEL cognitive processing script purpose.
6. Outline script and identify props, roles and entry condition requirements.
7. Flesh out the script outline by describing the different cognitive processing events or scenes that occur, from start to finish. Any type or combination of types of knowledge processing may be specified:
   a. Surface procedural knowledge
   b. "Deep" procedural knowledge
      1) quantitative knowledge processing
      2) qualitative knowledge processing
   c. Mental modeling
   d. Feature mapping to domain factual knowledge.
   e. Other
8. Identify and describe script execution results - driving actions and/or creating new internal symbol structures for driving future cognitive processing.
9. Where possible identify and describe sub-scripts and sub-script alternatives (tracks).

Motoric Section: (PUBLIC SCRIPTS FOR PUBLIC BEHAVIOR) (RESULTS: PUBLIC & PRIVATE)

10. Discuss KEL motoric activity script purpose.
11. Outline script and identify props, roles and entry condition requirements.
12. Flesh out the script outline by describing the different motor activities or scenes that occur, from start to finish.
13. Identify and describe script execution results - driving public actions and/or creating new internal private symbol structures for driving future cognitive processing.
14. Where possible identify and describe sub-scripts and sub-script alternatives (tracks).

Figure 7. KEL Object Template

REFERENCES

Anderson, J.R. (1990). *Cognitive Psychology - and Its Implications*. 3d ed. Freeman.

Booch, G. (1991). *Object Oriented Design with Applications*. Benjamin/Cummings, p. 35.

Bruner, J.S., Goodnow, J.J. and Austin, G.A. (1956). *A Study of Thinking*. Wiley.

Chase, W.G. and Simon, H.A. (1973). "The Mind's Eye in Chess," in W. G. Chase ed. *Visual Information Processing*. Academic Press.

de Groot, A. (1978). *Thought and Choice in Chess*. The Hague: Mouton.

Ericsson, K.A. and Simon, H.A. (1980). "Verbal Reports as Data," *Psychological Review*, 87, 3, 215-251.

Ericsson, K.A. and Simon, H.A. (1984). *Protocol Analysis - Verbal Reports as Data*. The MIT Press.

Ericsson, K.A. and Smith, J. (1991). "Prospects and Limits of the Empicical Study of Expertise: an Introduction," in K. A. Ericsson & J. Smith eds. *Toward a General Theory of Expertise: Prospects and Limits*. Cambridge University Press.

Gilmartin, A., Newell, A. and Simon, H.A. (1976). "A Program Modeling Short-term Memory Under Strategy Control." in C. N. Cofer ed. *The Structure of Human Memory*. San Francisco: Freeman.

Haney, W.V. (1992). *Communication and Interpersonal Relations: Text and Cases - 6th Edition*. Irwin.

Harmon, P. and King, D. (1985). *Expert Systems: Artificial Intelligence in Business*. Wiley.

Hart, A. (1986). *Knowledge Acquisition for Expert Systems*. McGraw-Hill.

Malandro, L.A., Barker, L. and Barker, D.A. (1989). *Nonverbal Communication: Second Edition*. McGraw-Hill.

Mayer, R.E. (1992). *Thinking, Problem Solving, Cognition: Second Edition*. W.H. Freeman.

Merkhofer, M.W. (1987). "Quantifying Judgmental Uncertainty: Methodology, Experiences, and Insights." *IEEE Transactions on Systems, Man, and Cybernetics, SMC-17*, 5, 741-752.

Miller, G.A. (1956). "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information." *Psychological Review*, 63, 2, 81-97.

Neches, R., Langley, P. and Klahr, D. (1987). "Learning, Development, and Production Systems." in D. Klahr, P. Langley & R. Neches eds. *Production System Models of Learning and Development*, The MIT Press, 35, 1987.

Newell, A. and Simon, H.A. (1972). *Human Problem solving*, Prentice-Hall.

Newell, A. and Simon, H.A. (1976). "Computer Science as Empirical Inquiry: Symbols and Search." *Communications of the ACM*, 19, 3, March.

Nisbett, R.E. and Wilson, T.D. (1977). "Telling More Than We Can Know: Verbal Reports on Mental Processes," *Psychological Review*, 84, 3, 231-259.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. (1991). *Object-Oriented Modeling and Design*. Prentice Hall.

Schank, R.C. and Abelson, R.P. (1977). *Scripts, Plans, Goals and Understanding: An Inquiry into Human Knowledge Structures*. Lawrence Erlbaum Associates.

Shephard, G.G. (1990a). *Guided Self-Elicitation of Decision Analyst Expertise*. Dissertation: Arizona State University.

Shephard, G.G. (1990b). "Developing a Guided Self-Elicitation Methodology for Natural Human Expertise," *Proceedings of DSI 1990 Annual Meeting*, San Diego, CA.

Shephard, G.G. (1991). "Guided Expert Self-Elicitation: Generalizing Captured Expertise," *Proceedings of Hawaii International Conference on System Sciences (HICSS-24)*, Computer Society Press.

Shephard, G.G. (in press). "Providing Descriptive Power to Guided Self-Elicitation," *Knowledge Acquisition*.

Simon, H.A. and Gilmartin, K. (1973). "A Simulation of Memory for Chess Positions," *Cognitive Psychology*, 8, 165-190.

Slatter, P.E. (1990). "Models of Expertise in Knowledge Engineering", in H. Adeli ed. *Knowledge Engineering - Volume I Fundamentals*, McGraw-Hill, pp. 130-156.

Spetzler, C.S. and Stael von Holstein, C.A. (1975). "Probability Encoding in Decision Analysis." *Management Science*, 22, 3, 340-358.

Stael von Holstein, C.A.S. and Matheson, J.E. (1978). *A Manual for Encoding Probability Distributions*. prepared for Defense Advanced Research Projects Agency under subcontract for Decisions and Designs, Inc., SRI International, Menlo Park, CA, Final Report, SRI Project 7078.

Steele, P., Murphy, J. and Russill, R. (1989). *It's a Deal: a Practical Negotiation Handbook*. McGraw-Hill.

Turban, E. (1992). *Expert Systems and Applied Artificial Intelligence*. Macmillan.

Tversky, A. and Kahneman, D. (1974). "Judgment Under Uncertainty: Heuristics and Biases," In D. Kahneman, P. Slovic & A. Tversky eds. *Judgment Under Uncertainty: Heuristics and Biases*, Cambridge University Press, 3-20, 1982.

Vandamme, F. (1987). "Knowledge Extraction from Experts in View of the Construction of Expert Systems", in J.L. Mumpower, L.D. Phillips, O. Renn and V.R.R. Uppuluri eds. *Expert Judgment and Expert Systems*, Proceedings of the NATO Advanced Research Workshop on Expert Judgment and Expert Systems held in Porto, Portugal, August 25-29, 1986, pp. 321-337, Springer-Verlag: Berlin Heidelberg New York.

Zechmeister, E.B. and Nyberg, S.E. (1982). *Human Memory*. Brooks/Cole Publishing (Wadsworth), Belmont, CA.

MODELING in EMA

Susan Spirgi, Dieter Wenger
Swiss Bank Corporation, Hochstrasse 16, 4002 Basel, Switzerland
Email: wbs@clients.switch.ch

ABSTRACT

EMA is a methodology supporting model-based prototyping for developing knowledge-based applications. The modeling in EMA is its characteristic part. Only one model exists, structured according to a formalism called the meta-model. The meta-model consists of the basic meta-concepts *concept* and *relation*. Further meta-concepts are kinds of these basic meta-concepts. Modeling in EMA is divided into concept modeling and transformation modeling. A central concept is the *process*. It helps to derive which information must be kept in the model and which is general. *Areas* are introduced for modeling the environment which is necessary for modeling the interaction of the application with its environment. Modeling the transformation leads to the meta-concept *agent* which performs a transformation. An agent accesses information and affects its environment by *effects*. The meta-concept effect is central and characteristic for modeling, because the state of an application is exclusively changed by effects.

1. INTRODUCTION

This paper discusses the modeling in EMA. An introduction and overview of EMA and its positioning is presented in the related paper (Spirgi and Wenger, 1992). Readers not familiar with the EMA approach are recommended to read the related paper first.

EMA consists of one model which contains the specification of the application. The formalism of the model which is denoted in EMA as the meta-model, consists of meta-concepts which are introduced in chapter 2. Chapter 3 discusses *concept modeling*. The relationship and transformation between different representations are pointed out, such as lists and the relational model. Chapter 4 discusses the two concepts *process* and *knowledge*. It is an introduction for *transformation modeling* and leads then to the concept *agent*. The final chapter gives an idea of the way agents are developed.

2. THE BASIC META-CONCEPTS: CONCEPT AND RELATION

The domain which has to be modeled is *information processing*. It is assumed that the world of information processing is built on the two basic and atomic meta-concepts *concept* and *relation*. The meta-concept concept has two types:

- object
- attribute

The meta-concept relation has following four types:
- association
- object-attribute relation
- abstraction
- transformation

The relations abstraction and association relate concepts. The meta-concepts concept, association, object-attribute relation and abstraction are the meta-concepts for *concept modeling*.

The meta-concept transformation relates *information* which is composed of concepts. The transformation transforms information into new information. Similar kinds of pieces of information which are denoted as *information-units* (info-unit) are comprised to *information-types* (or also called *information-sets*). E.g. the object 'car' with its attribute 'color' builds an information-type. The name composed of 'car' and 'color' is the name of the set (car@color) which comprises all the info-units which are built by the instances of 'car' and by the inherited attribute 'color'. A transformation is defined by specifying all the *causal relations* (Figure 1). A causal relation relates information-units of the input information-sets to information-units of the output information-sets.

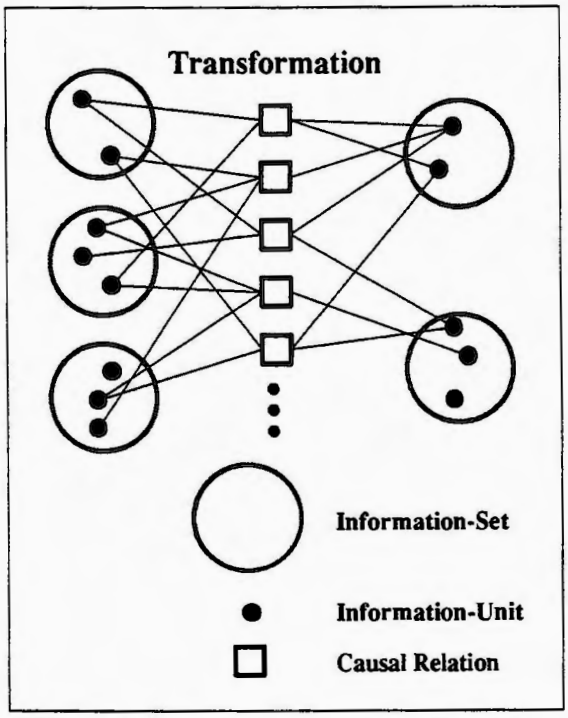


Figure 1 The meta-concept transformation

Information and transformation are the meta-concepts for *transformation modeling*. A transformation will be specified later and will then lead to the concept *agent*.

In EMA the formalism itself is seen as a model which models the model and which is therefore called meta-model. This is in analogy to the term meta-database in the domain of database technology. It is therefore logical that the meta-model is modeled with the same formalism which means with itself. Thus, there are objects such as 'object', 'attribute', 'agent'. Examples which are used in the next chapter are representations of the meta-model of EMA.

3. CONCEPT MODELING

Concepts can be modeled in following ways:
- in objects (frames, schemata, etc.)
- in tables
- in lists

Each kind of modeling has its strengths and drawbacks. A formalism is sought which should be the base for all kinds of concept modeling. Figure 2 shows that these kinds are no longer seen as autonomous modeling formalisms. They are views (representations) of a more general modeling formalism. This new role of the different concept modeling approaches serves to achieve two objectives - a conceptual and a technical one.

The conceptual objective is the representation of the model. Each of these views has its strengths representing and emphasizing certain aspects of the model. The technical objective is an optimal technical management of the concepts. Depending on the concepts, parts of the model are kept in objects, tables or lists.

Figure 2 shows that there is one *concept view* of the model, that this concept view can be represented by many different representations and that all these representations are views of the same model.

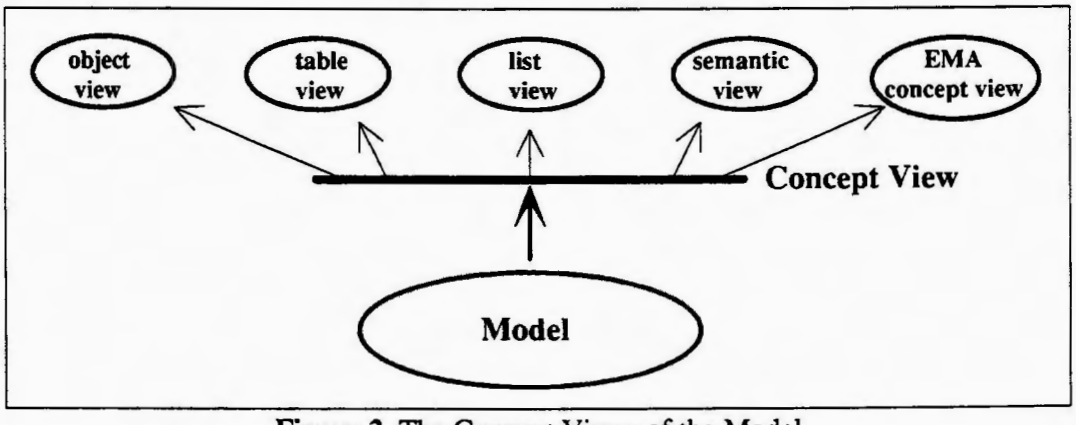


Figure 2 The Concept Views of the Model

As mentioned in chapter 2 the formalism for modeling concepts contains the following meta-concepts:

- concept
  - object
  - attribute
    - value-attribute
    - method-attribute
    - association-attribute
- relation
  - association
  - object-attribute relation
  - abstract relation
    - is-a (ia)
    - is-role-of (ir)
    - instance-of (iof)
    - is-part-of (ip)

The *EMA concept view* has among these views a special position, because it has been especially developed for representing the model. Therefore, the EMA concept view is normally used for representing the concept view of the model. Figure 3 shows the symbols for the meta-concepts.

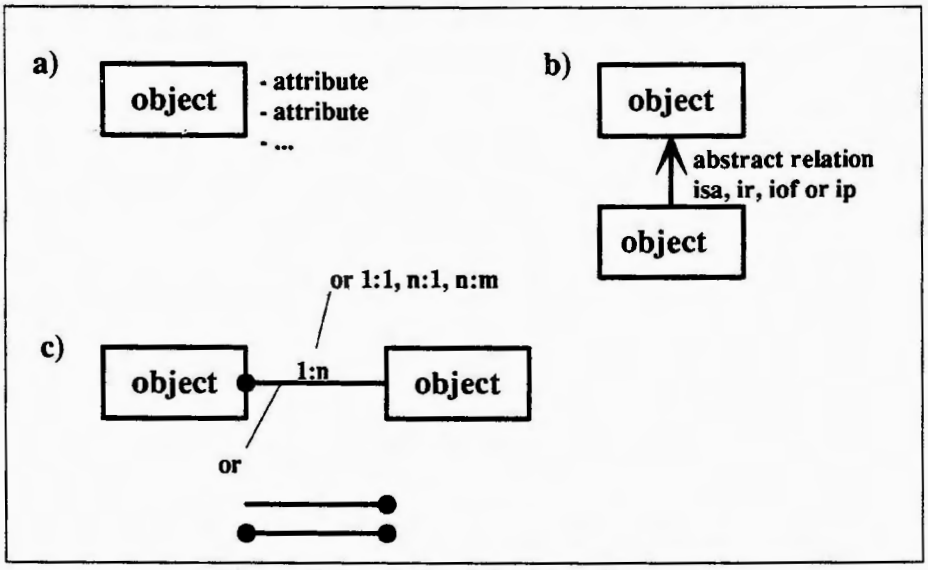


Figure 3 Symbols of the Meta-Concepts

Figure 3a symbolizes an object and attributes which are related by the object-attribute relation, Figure 3b an abstract relation between two objects and Figure 3c the assocation between two objects. A filled circle indicates which object carries the association.

The views serve as the developer interface which facilitates the communication between developer and model.

In chapter 3.1. and 3.2. the significance of the list and table view is outlined, along with the way the views are obtained from the EMA concept view and hence from the model.

3.1. The List View

Representing concepts with lists is extremely powerful in the case of complex structures. The characteristics of a complex structure is that it consists of a large amount of relations. Pure information can be related without help information, such as references or identifications.

With the help of an example, represented as an EMA concept view (Figure 4), it is shown how the list view is derived. At the moment, the meaning of this view is not of interest. Later it will be shown that it represents a part of the EMA formalism itself (meta-model).

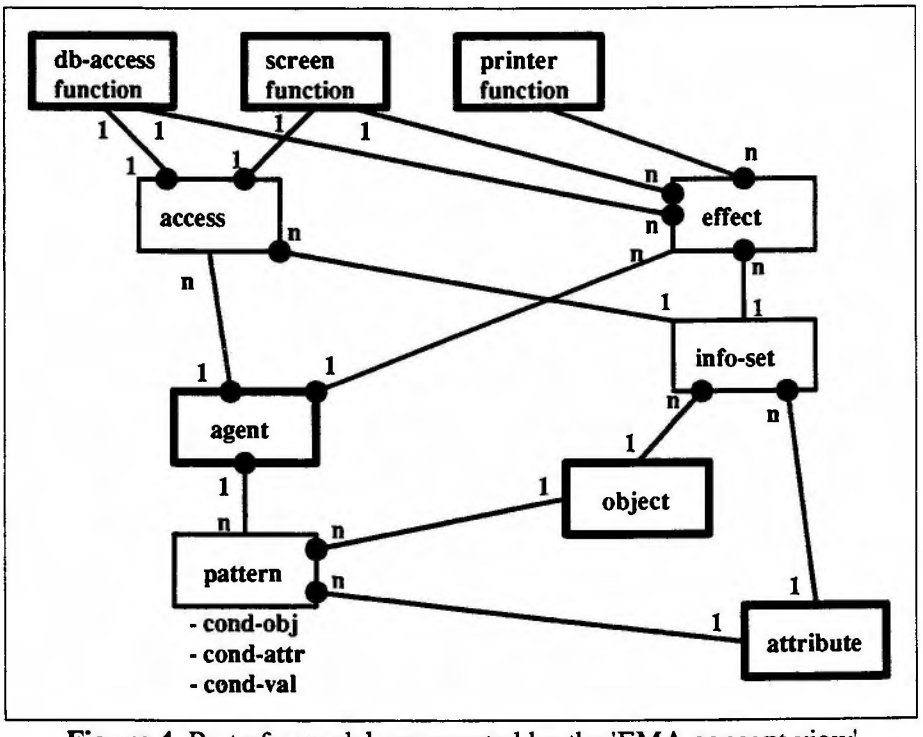


Figure 4 Part of a model represented by the 'EMA concept view'

Figure 4 contains the two kinds of objects: one with an identification called *key object* (bold) and one without which is called *relational object*. Figure 5 helps to explain the relational object. It shows the modeling of the object-attribute relation which is a part of the meta-model. The association between the two meta-concepts object and attribute is represented. Between both of them a n:m association exists. The problem is now that this association has the attributes 'value' and 'descr'. Therefore a relational object must be defined which shall be called 'obj-attr-rel', and which has the attributes 'value' and 'descr'. The difference to an object, such as 'object' or 'attribute', is that the instances of the relational object have no identification. They only have attributes.

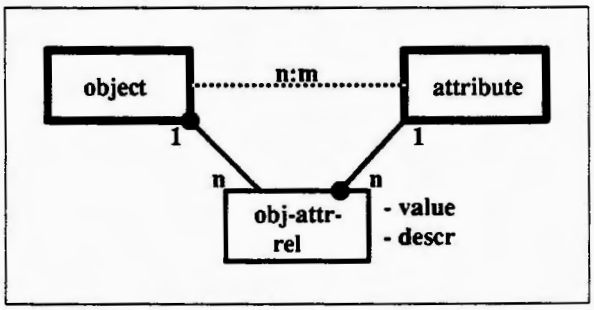


Figure 5 Relational and non-relational object

In Figure 4 'agent' is the central object. The goal is to put the structure around 'agent' in list views. Each association from 'agent' to any other object leads to one list view. The guidelines are applied for the association between 'agent' and 'pattern'. The association between 'agent' and 'pattern' gives us the first term

agent-pattern

which is in this case the default name of this association. Each association may have an explicit name. If this explicit name exists, then it is taken instead of the default name. In the case of a 1:n association the next step is to open a parenthesis.

agent-pattern (

Then the object name is added as a variable consisting of a '?' and the name of the object (?pattern), but because 'pattern' is a relational object no variable is added (the variable stands for the identification). Then each of the attributes provides a name and a variable.

```
agent-pattern (cond-obj ?cond-obj
               cond-attr ?cond-attr
               cond-val ?cond-val
```

Then next step is to add the associations (they have in this case their default names) of the relational object 'pattern'.

```
agent-pattern (cond-obj ?cond-obj
               cond-attr ?cond-attr
               cond-val ?cond-val
               pattern-object ?object
               pattern-attribute ?attribute)
```

No preceding parenthesis is inserted, because we have n:1-associations. Then next step would be to add the object associated with 'pattern'. But we stop, because we come to key objects.

The same guideline leads to the following two other list views:

agent-effect ***(effect-infoset*** ***infoset-object ?object***
infoset-attribute ?attribute
effect-screen-fct ?screen-fct
effect-print-fct ?print-fct
effect-db-fct ?db-fct)

agent-access ***(access-infoset*** ***infoset-object ?object***
infoset-attribute ?attribute
access-screen-fct ?screen-fct
access-db-fct ?db-fct)

These list views can be strongly reduced, because only the variables must be kept in a list view. The other terms are keywords which are not kept in a list view, because their position indicates implicitly the kind of information. Erasing all of the keywords results in the list views:

agent-pattern (?cond-obj ?cond-attr ?cond-val ?object ?attribute)

agent-effect (?object ?attribute ?screen-fct ?print-fct ?db-fct)

agent-effect (?object ?attribute ?screen-fct ?db-fct)

These list views are not the concrete lists but their descriptions. The concrete lists are built according to the list views. Each expression in parentheses can be repeated n times. As an example, the list view (?a ?b (?c ?d (?e) ?f)) can result in a list such as:

```
(     (a1 b1   (c1 d1  (e1 e2 e3) f1)
               (c2 d2  (e4 e5) f2)
      )
      (a2 b2   (c3 d3  (e6 e7 e8) f3)
               (c4 d4  (e9) f4)
               (c5 d5  (e10 e11) f5)
      )
)
```

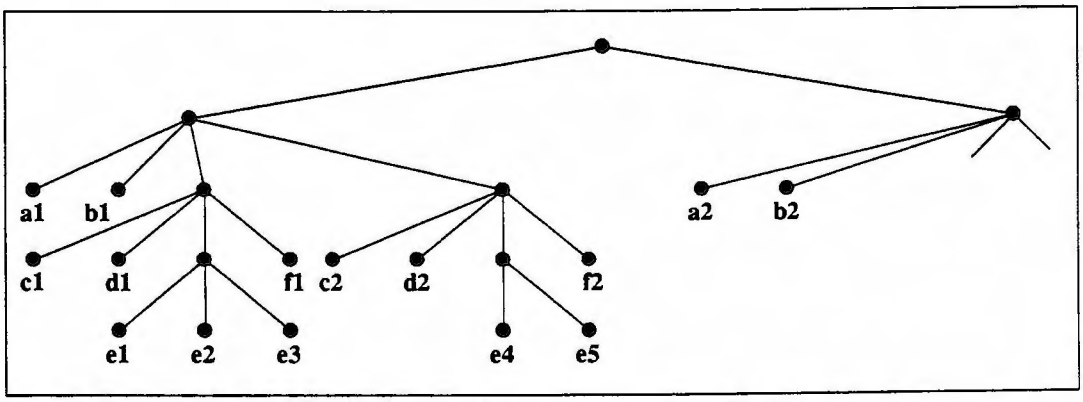


Figure 6 Graphical representation of a list

This list is partly graphically represented in Figure 6.

The first term (agent-pattern, agent-effect, agent-access) of a list view is seen as its name. The lists are combined with objects as follows: the name of the list becomes to the attribute and the list to the value of the attribute. In the case of our example we get the combination represented in Figure 7.

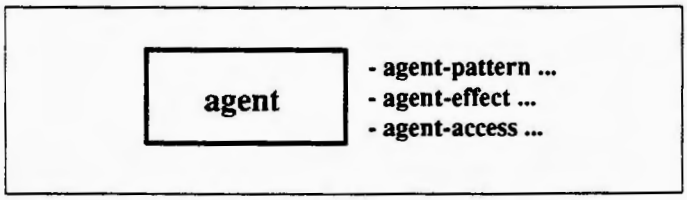


Figure 7 Interface between object and list

3.2. The Table View

From the conceptual standpoint the representation of concepts with tables is appropriate, if many objects are related by many associations. From the technical standpoint concepts with huge amounts of instantiations can be kept very efficiently in tables.

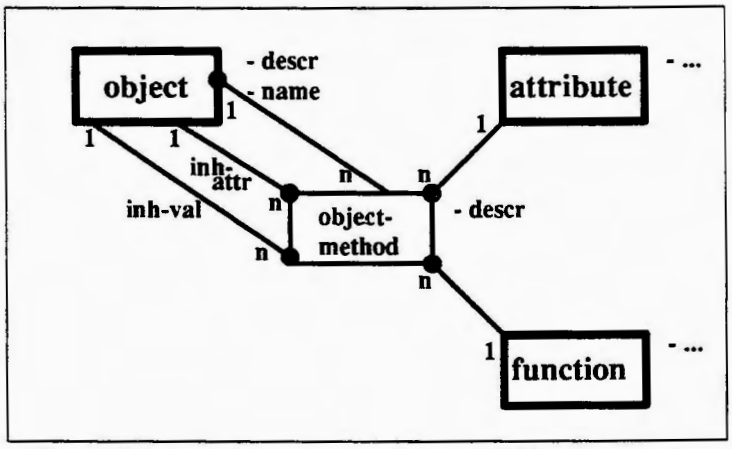


Figure 8 Example represented as an EMA concept view

Assume the list view (a b (c d (e) f)) of the previous chapter. As soon as we have thousands of repetitions, the list is no longer an adequate technical representation. One reason is the absence of auxiliary information such as references. In the previous chapter the same reason has been mentioned as a virtue, but for huge amount of information this plus becomes to a minus, because accessing elements of the list becomes too time consuming.

Figure 8 shows an example which represents the attributes which are methods (method-attribute) belonging to an object. This representation will be transformed into a table view (Figure 9) following certain guidelines. Each object results in a table and each attribute results in a column. In addition, the name of a key object becomes the key of its table. This way, we obtain table 'object', 'attribute' and 'function'. Next step is to build table 'object-method' which is placed one layer lower than the table 'object', because the quantitative relationship of the association between 'object' and 'object-method' is 1:n. All the associated

key objects are transformed to a column, in addition the attributes are transformed to a column, too.

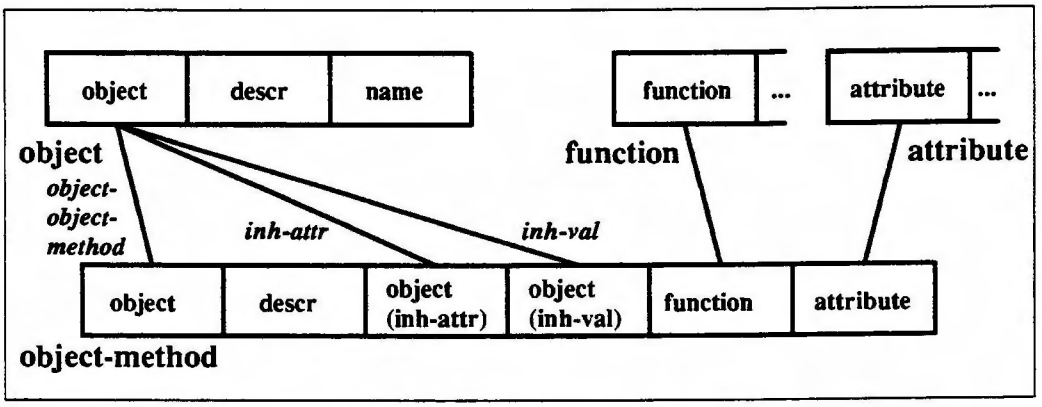


Figure 9 Table view of Figure 8

More than one association may exist between two tables (as also between objects). As an example we have the following associations between the tables 'object' and 'object-method': an object contains methods (association 'object-object-method'); a method-attribute of an object is associated with the object from which this attribute is inherited (association 'inh-attr'); and the same for its value (association 'inh-val').

This way, a table view is obtained which is called in the data-base domain 'relational model'. Moreover, semantic net representations could also be constructed.

4. PROCESS AND KNOWLEDGE

In EMA the concept *process* plays an important role especially in respect to transformation modeling. Therefore, this concept is outlined in detail.

Figure 10 shows the components of a process which are input, output, processor, knowledge- and data-information.

Input information is processed to output information driven by the knowledge-information. A computer application e.g. can be seen as a process, where the code is the knowledge-information and the computer-processor the processor. By changing the knowledge-information (code), the computer application becomes different. Changing the processor is not the usual way of changing an application. The processor is the hard, the knowledge-information the soft component of a process.

Another example of a process is the binary OR-operator in Figure 11. Assuming no knowledge-information (table in Figure 11) exists, then the processor itself knows how to do an OR-operation. However, this processor is specialized in OR-operations and is not able to do any other binary logical operation. This process has no flexibility and the generality is 0. If we do have knowledge-information such as the table in Figure 11, then we can do any binary logical operation, depending on the contents of the table. If the values for input and output are either 0 or 1 and exactly 2 inputs and 1 output are given, then the processor is 100% general,

because by changing the knowledge-information all possible characteristics (16 in total) can be achieved.

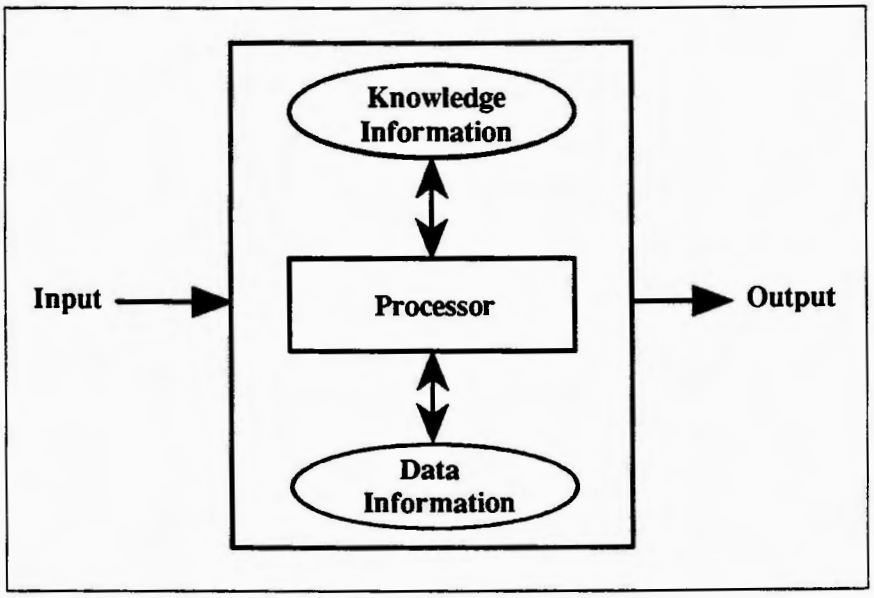


Figure 10 Components of a processor

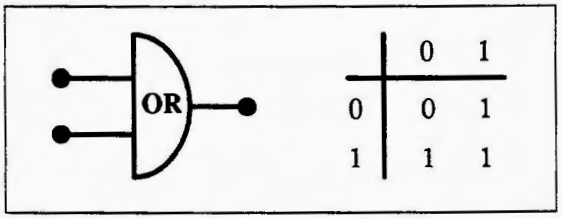


Figure 11 Binary OR-Operator

The knowledge-information is the knowledge of the process which drives the processor. But there must also be another kind of knowledge incorporated in the processor, because the processor has to know how to work according to the knowledge of the process (interpreting the knowledge of the process). This can be also seen, if we take our binary processor with the knowledge-information for an OR-operation and create an OR-processor without knowledge-information by bringing the knowledge (information of the table) into the processor. This way, knowledge for driving the processor is brought into the processor. By integrating knowledge of the process into the processor, the process becomes more specialized. The process becomes less general but more efficient. Another reason that a processor contains knowledge is that a processor is a process which is again consisting of the components of a process. In EMA, the fuzzy term knowledge is information which drives a processor. Information is a term which is exactly defined by meta-concepts of EMA (see chapter 5.2.) and thus knowledge gets a clear meaning, although limited to the EMA approach.

In EMA, a knowledge-based application is seen as a process. In (Spirgi and Wenger, 1992) it is pointed out that the knowledge-information of a knowledge-based application is generated from the information stored in the model. This model is managed by the EMA methodology-

application which models the application and which is itself a process. The model is the data-information of the EMA process. Thus, three levels where knowledge is located can be identified: the knowledge of the data-information of the EMA process on the *specification level*; the knowledge of the knowledge-information of the application on the *execution level*; the knowledge of the processor of the application on the *processor level*. The two last mentioned levels are comprised to the *processing level*. The central question for the development of a methodology is, what kind of knowledge should be on the specification, what on the execution and what on the processor level. The knowledge on the specification level is stored according to a formalism. The knowledge on the execution level is generated of the knowledge of the specification level which is interpreted by the processor on the processor level. Therefore, between these three levels a concordance is needed.

Another important aspect for the development of a methodology is: the better ability of putting knowledge into an application the better applications can be developed. A major part of the knowledge within an application is general knowledge. The knowledge of how an addition is performed e.g. is not application-specific, but general knowledge. It seems to be obvious that application-specific knowledge is on the specification level and general knowledge on the processing level. A first example seems to confirm that. Assume *function* is a meta-concept of the modeling formalism (actually it is). An *elementary function* like addition has no application-specific knowledge. However, the information of how the addition is applied is application-specific. Therefore, the information for applying functions is stored in the model, the information how functions are carried out is part of the processor. Thus, elementary functions are only found on the processing level.

A second example refutes the assumption. A problem-solving technique e.g. the A* search algorithm is partly general and partly application-specific. In order to integrate the technique, the application-specific part has to be developed. For this, the knowledge of how this technique works must exist. This means, the part of the knowledge which is specific to the problem-solving technique must be on the specification level, although this knowledge is general. Another aspect is that the processor which is part of each process must not become too large. Thus e.g. a technique as the A* ought not to be part of the processor, because the A* is very rarely used. A piece of knowledge ought to be carried on the specification level, although it is general.

The challenge is therefore to find the most extensive non reducible building-blocks which are completely general and which cover the most of the general knowledge but nothing more. These building-blocks are the elementary meta-concepts of a methodology.

All this leads us to the following principle:
The interface between the knowledge-information and the processor consists of the *elementary meta-concepts*. They are elementary knowledge building-blocks which have a defined meaning and are directly interpreted by the processor according to their meaning. Among them are some of the meta-concepts for concept modeling (concept, abstract-relation, etc.) and the elementary functions such as addition. These elementary meta-concepts are the elementary building-blocks for modeling knowledge. Therefore, the knowledge which can be modeled is determined by the elementary meta-concepts. The knowledge on the specification level, either general or application-specific, is built with these elementary and *high-level meta-concepts*. High-level meta-concepts are built with high-level and elementary meta-

concepts, where the high-level meta-concepts consist ultimately of the elementary meta-concepts. The high-level meta-concepts are problem-solving building-blocks which are called G-TECs in EMA (Spirgi, Wenger, and Probst, 1991).

This process approach should show that knowledge on the knowledge and processing level is strongly related and that therefore the cooperation and not their segregation should be emphasized. Figure 12 shows that the application-specific knowledge and the required general knowledge, both on the specification level, build together the knowledge-information of the process.

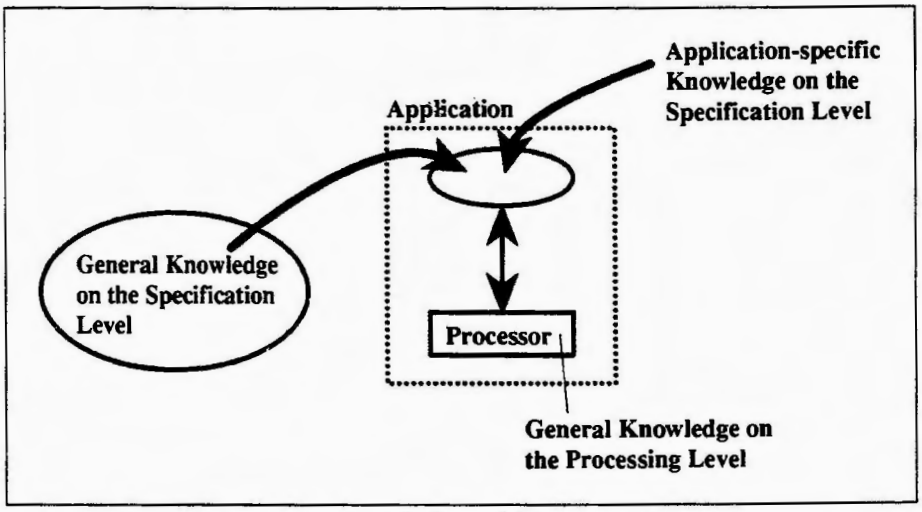


Figure 12 Knowledge within an application

5. TRANSFORMATION MODELING

Principally a transformation transforms input- to output-information. In order to perform this, a transformation has to access and deliver information from and to somewhere. Information can be accessed from the screen, the data-base, etc. Similarly information can be sent to the screen, the printer, the data-base, etc. A transaction to access information shall be called *access* and to send an information, *effect*. Screen, data-base, printer, etc. are called information areas. This way, an access accesses the information area and an effect affects it.

5.1. Information Areas

Based on the approach that a knowledge-based application is seen as a process consisting of input, output, a processor, knowledge-information and data-information (Figure 10 and outlined in (Spirgi and Wenger, 1992) in more detail), one internal and a number of external information areas can be identified. The knowledge and data-information of the process build the *internal information area*. External areas are outside the process and provide the input for the process or are the areas where the output goes.

In each of these areas concepts with their relationships exist. Therefore, these areas are modeled by concept modeling. Parts of the concept models of the areas are accessed and

affected. These accessed and affected parts are defined as *information*. The concept information is thus strongly related to the accesses or the effects which are performed by functions called either *access-* or *effect-functions*. The function accessing or affecting the smallest no further divisible piece of information is called an *elementary function*. The database area e.g. is modeled with the objects 'table' and 'attribute'. The information which can be accessed is for example the attribute of a table. For doing this an access-function is needed.

5.2. The Internal Information Area

The internal information area is the central working area of the process and is built according to the meta-model for concept modeling. Accessed and affected are the attributes of objects, the relations between objects and the objects themselves. Therefore information consists of:

- object @ attribute @ value
- object @ relation @ related object
- object

These three expressions are the descriptions for information of the internal information area. If such a description specifies exactly one information, then it describes one *info-unit*, otherwise it specifies a set of info-units which is called an *info-set*. The following example should clarify the terms info-set and info-unit.

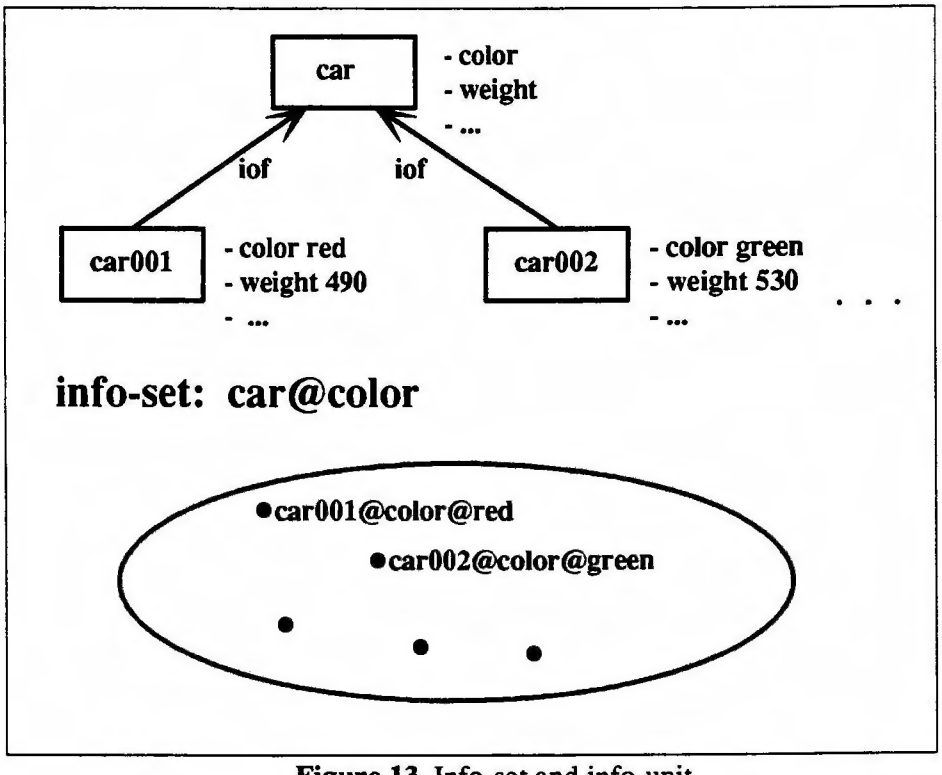


Figure 13 Info-set and info-unit

An info-unit is an atomic piece of information and consists of an object, one of its attributes and the value (object@attribute@value). An info-set consists of an object and one of its attributes (object@attribute) and is a set of all the info-units of which the objects are in an is-

a, instance-of or is-role-of relation to the same object and have the same attribute (inherited attribute). An info-set comprises information of the same type. In Figure 13 the info-units car001@color@red (color red of car car001) and car002@color@green belong to the info-set car@color.

6. THE AGENT: DATA-DRIVEN, OBJECT- AND EFFECT-ORIENTED

A transformation is performed by a concept which shall be called agent.

6.1. Structure of an Agent

An agent consists of an *activation*, a *retrieval* and a *processing part* (see Figure 14). The processing part produces effects which change the state of the application. The retrieval part accesses all the information which is processed by methods (functions), in order to set up the necessary effects. All areas can be accessed. The activation part is responsible for the activation of the agent. A match is done comparing the internal information area with the pattern. The activation pattern, the retrievals and the effects characterize the agent, although not exhaustively.

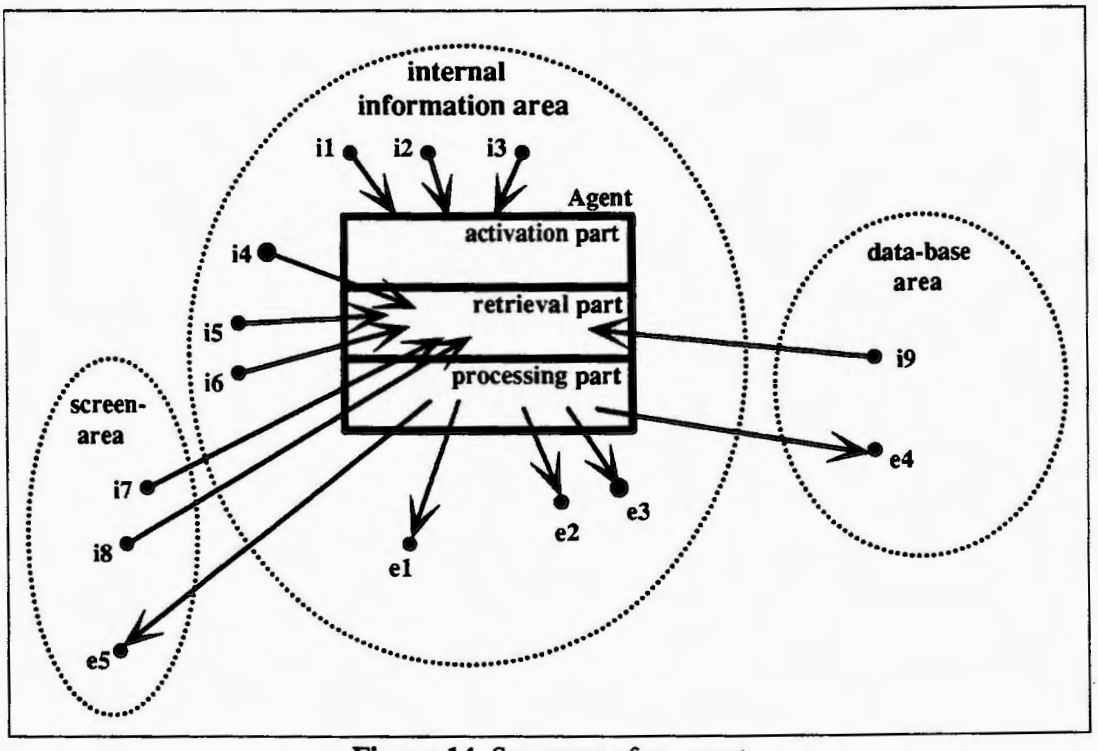


Figure 14 Structure of an agent

In the example of Figure 14 the pattern of the activation part matches the information i1, i2 and i3. The agent accesses the internal information area (i4, i5 and i6), the screen area (i7, i8) and the database area (i9). The effects, produced by the processing part manipulate the internal information area (e1, e2 and e3), the database area (e4) and the screen area (e5).

The three different parts of an agent have their specific non overlapping tasks. The activation part is responsible for the activation of an agent, the retrieval part for getting all the necessary information and the processing part for the processing and the effects.

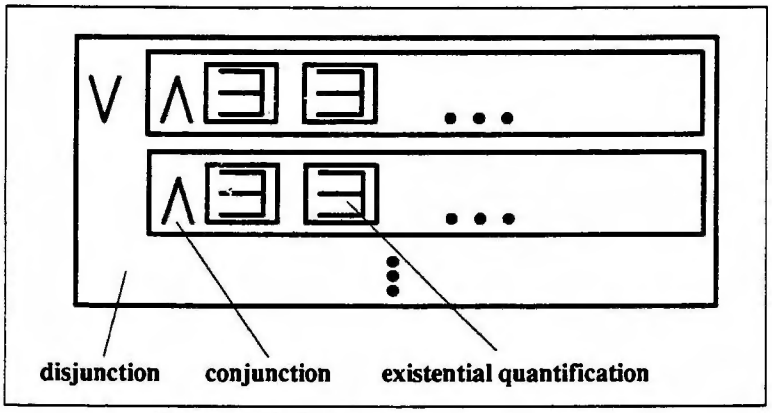


Figure 15 Activation part of an agent

6.2. Activation Part

The activation part consists of a *pattern* which is looked for in the internal information area. As soon as there is a match between this pattern and information of the internal information area, the agent is activated. The pattern is built up by a disjunction of conjunctions (Figure 15). The elements of a conjunction are existential quantifications of predicates.

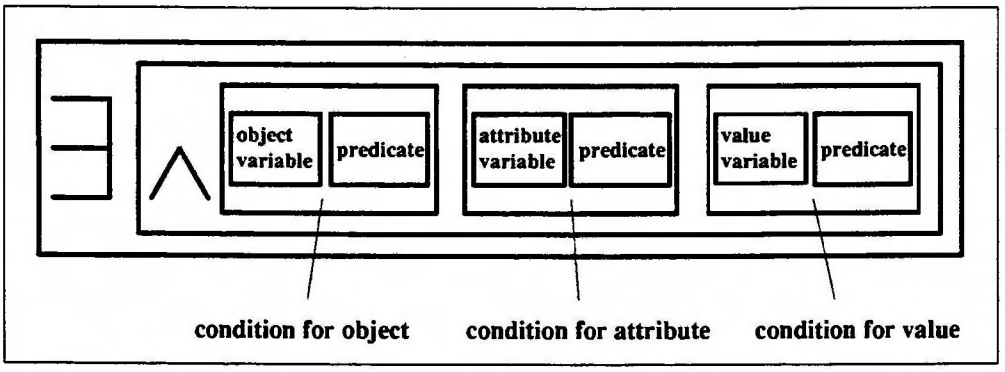


Figure 16 Structure of an existential quantification

The existential quantification tries to match at least one info-unit in the internal information area. The structure of an existential quantification can be seen in Figure 16. In the object part (or attribute part, or value part) the object variable can be bound to the object (or attribute, or value). Any bound variable may be used in any subsequent predicate. Figure 17 shows two examples. In one example the agent should trigger if at least one red car exists. In the other example the agent should trigger if at least one red car with a weight of less than 500 kg exists.

All the bound variables are available for the retrieval and the processing part.

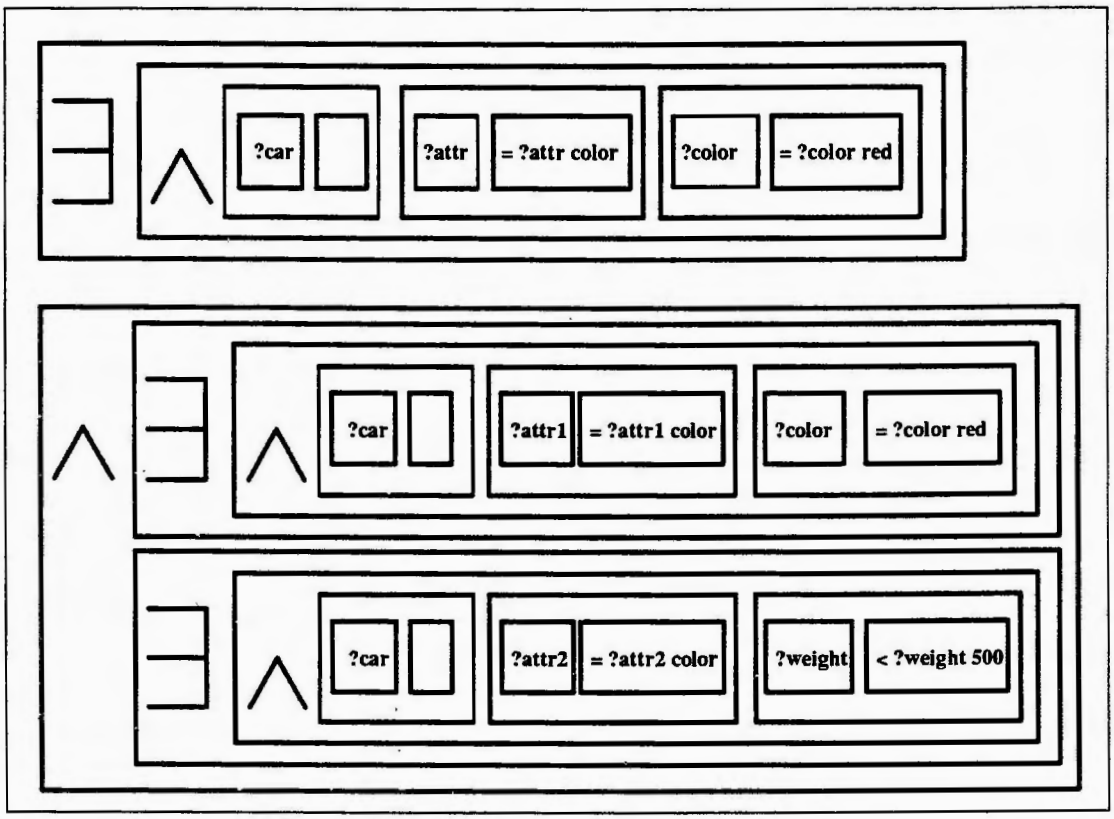


Figure 17 Two examples for the activation part

6.3. Retrieval Part

The retrieval part is separated from the processing part, because the former acts as an interface between the areas and the local data of an agent and consists of access-functions which can be classified according to the kind of accessed areas and according to the kind of access. Data accessed by the access-functions is bound to variables which are available for the processing part.

6.4. Processing Part

The goal of an agent is to generate effects, because only effects change the state of an application. The job of the processing part is to prepare the information which is used by the effect-functions for producing effects. The effect-functions can be classified according to the area they affect (manipulate) and the kind of effect they produce. The entire processing part is submitted to produce effects. Therefore, the effects are used for structuring the processing part.

Figure 18 shows that at the beginning of the processing the bound local variables of the activation (a1, a2) and of the retrieval part (r1, r2, r3, r4, r5) are available for processing. They build the input data. The data in r2, r3 and r5 are accessed from the internal information area, whereas r1 and r4 come from an external area (e.g. data-base area).

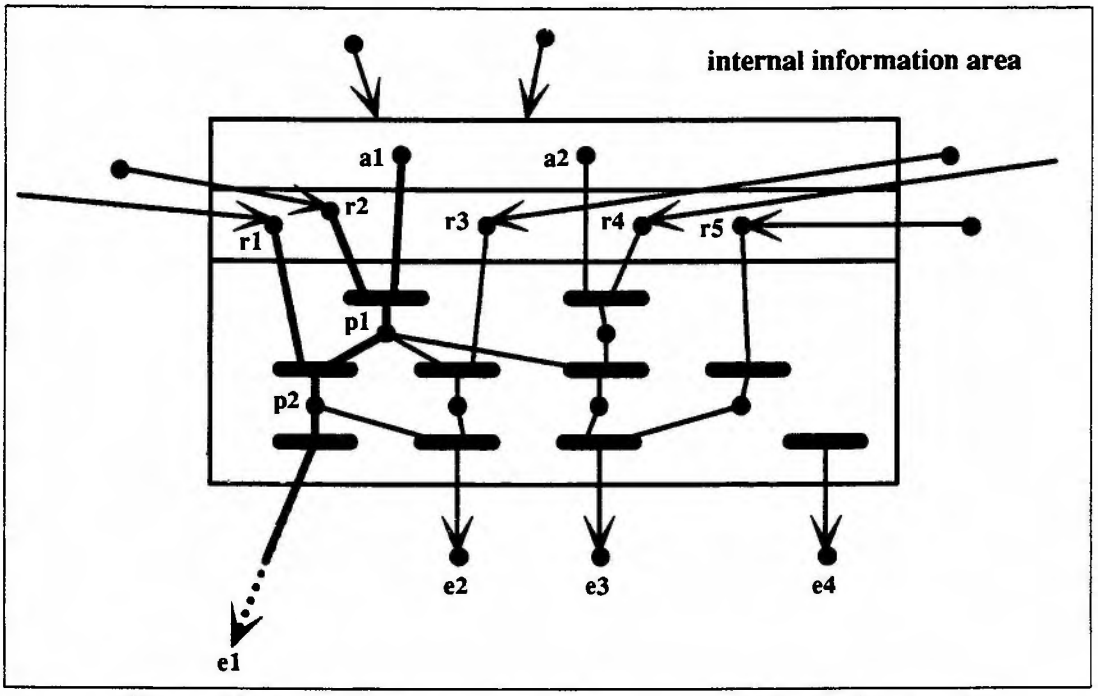


Figure 18 Example of an agent with the structure of the retrieval part

The data bound by these variables are processed by functions to new data and bound in local variables of the processing part. Each effect has its variables and functions (e.g. see effect e1). Some of the variables can belong to more than one effect. The effects e2, e3 and e4 manipulate information in the internal information area, the effect e1 acts on any other area. It is important to know whether effects can be decoupled from each other. Assume the same effect has to be produced in more than one agent, then this effect is taken out of the agent and put into a new agent, because processing redundancy should be eliminated.

6.5. Functions within an Agent

In an agent the 4 different kinds of functions - *predicate*, *access-function*, *result-function* and *effect-function* - are used. Predicates return true or false and have no effects. Access-functions are the functions of the retrieval part. Result-functions return a piece of information and have no effects (a predicate is therefore a specialisation of a result-function). The effect-functions are the functions of the processing part.

The following table relates the different kinds of functions to the agent's parts:

predicate	-	activation, processing part
access-function	-	retrieval part
result-function	-	processing part
effect-function	-	processing part

The *elementary functions* are general and are directly interpreted by the processor. They can be assigned to exactly one type of function. Application-specific and general non-elementary

functions can be assigned to more than one type depending on the type of elementary functions they consist of.

6.6. The Meta-Model for Agents

In (Spirgi and Wenger, 1992) it is shown how the EMA methodology application is developed. The EMA methodology is the knowledge of the knowledge-based methodology application EMA. This knowledge is modeled with EMA itself. Thus, the meta-model is simply a model. Part of this model which models a part of the agent is represented in Figure 4.

7. DEVELOPING AGENT-BASED PROCESSING

An agent can be developed by starting with the activation (activation-oriented) or with the effects (effect-oriented). The two questions are:
- What activates the agent?
- What effects have to be produced?

The first question can be generalised to:
- What activates the various kinds of processing?

The development of agents is only touched in this chapter to give a rough idea. To cover all aspects other parts of EMA must be involved. Agents comprise processing which is structured by the effects. The question is:
- Which effects are assigned to which agents?

The further questions are:
- What are the effects?
- What are the activations?

Figure 19 helps to answer parts of these questions.

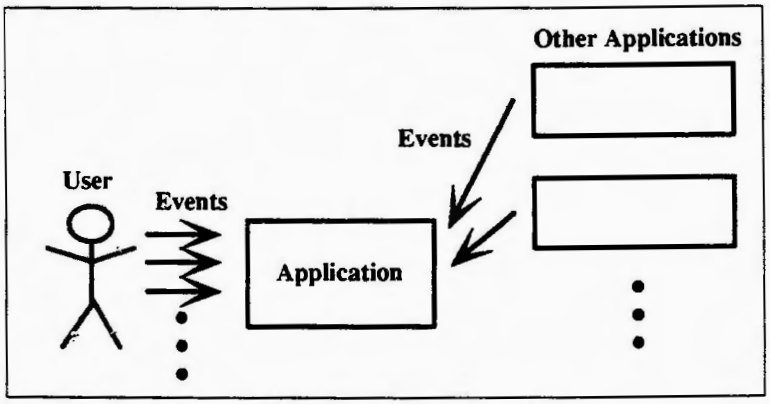


Figure 19 Context of an application

Two kinds of external events exist: events coming from the user and coming from other applications. Each event activates one agent. Therefore, at the beginning as many agents as events exist (Figure 19).

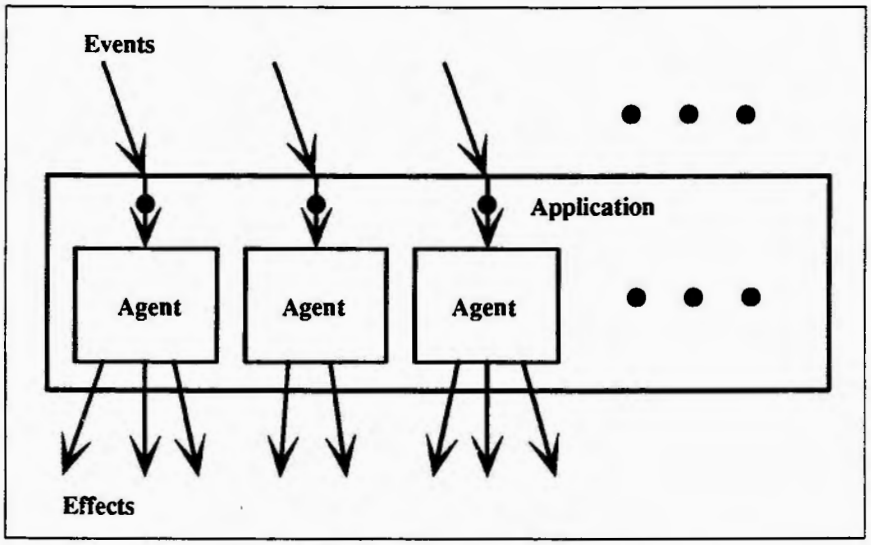


Figure 20 Events triggering agents

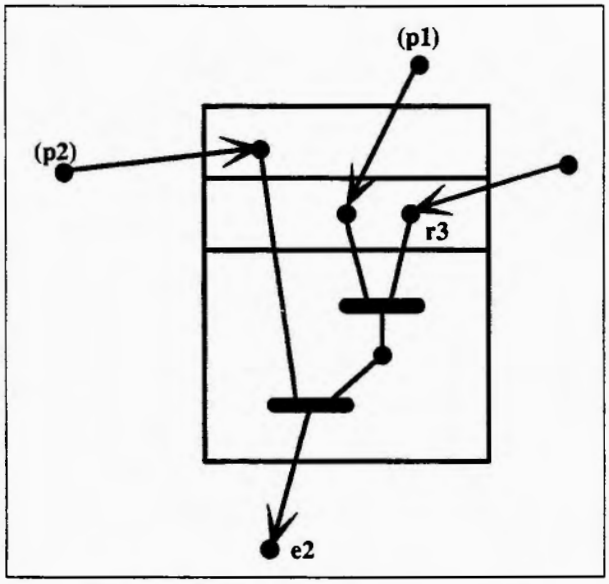


Figure 21 Creation of an new agent

But this cannot be the final state of an application, because (1) identical and similar effects are produced by more than one agent and (2) some intermediate processing of the agents is redundant. The criterion of non-redundant processing is the one which helps us to structure the application into agents. Parts of the processing of the existing agents are put into new agents. The result is that an increasing part of the local data within the agents becomes data in the internal information area. Assuming that the effect e2 in Figure 18 should be put into a new agent, then the local data p1 and p2 become data of the internal information area (Figure

21). Because p2 depends on p1, we know that if p2 is calculated, p1 has already been calculated and therefore p2 is used for the activation part.

8. DISCUSSION AND CONCLUDING COMMENTS

This modeling approach is compact and provides a complete model which is executable by transforming it into an interpretable formalism. Therefore, it allows a powerful prototyping which means that the EMA methodology application runs in parallel with the application in development. A modification in the model is immediately transferred to the application.

The most serious deficiency is currently the developing of the processing part of an agent. It is done by means of a structured editor based on a type of representation similar to Figure 18 and 21. This is not a satisfying situation, because the developer is not sufficiently supported. The structured editor is a too passive tool. Moreover, the kind of specification for this part is based on inadequate building-blocks, because e.g. redundancy is not prevented. To overcome these limitations, a way which can be described as *knowledge-based functional programming* is a main focus of our current and future work. Programming is seen as a task which is completely determined in the sense that no magic steps are carried out. Starting from an effect and its related accesses and based on four functional primitives (transformation, calculation, extraction and build) the task will be performed in cooperation by the developer and EMA.

REFERENCES

Spirgi S., Wenger D., Probst A. (1991). "Generic Techniques in EMA: A Model-Based Approach for Knowledge Acquisition", Proceedings of the 6th AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, KAW-91 Banff, pp 31-1-31-13

Spirgi S., Wenger D.(1992). "The Knowledge-Based Methodology-Application EMA", Proceedings of the 7th AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, KAW-92 Banff

The Knowledge-Based Methodology-Application EMA

Susan Spirgi, Dieter Wenger
Swiss Bank Corporation, Hochstrasse 16, 4002 Basel, Switzerland
Email: wbs@clients.switch.ch

ABSTRACT

EMA is on the one hand a methodology for developing any kind of knowledge-based application and on the other hand a knowledge-based application itself, where the methodology is the knowledge within this knowledge-based application. The central part of EMA is its model. This model stores all the information for application development which is the specification for the application. This approach can be called model-based prototyping, because the methodology-application and the application which is in development run in parallel. A modification which is made in the model is immediately transferred to the application. The final goal of EMA is, that the developer assisted by EMA specifies the application and that EMA develops the application according to the specifications.

1. INTRODUCTION

This paper presents an overview and introduction to the EMA methodology. It presents the new EMA approach which is still strongly based on the old one (Spirgi, Probst, and Wenger, 1990). The objective is, that the reader gets an insight of EMA. Two other papers are related to this overview, the one which presents the modeling in EMA (Spirgi and Wenger, 1992), the other which gives a detailed insight into the model-based approach of EMA (Spirgi, Probst, and Wenger, 1991).

Chapter 2 discusses the principles underlying this approach. Chapter 3 outlines the realisation of this methodology as a knowledge-based application. This chapter should also help to clarifiy the meaning of the title. The central chapter 4 is dedicated to the development of the methodology. The final chapter relates EMA concepts to those of other methodology approaches which are KADS, PROTÉGÉ and SBF.

2. PRINCIPLES OF EMA

A methodology for knowledge-based applications is seen as the knowledge for applying AI in order to build applications. It is a vehicle for achieving a goal. The developer is the driver who has to provide information and to control the application development, while EMA is executing the application development. Therefore, it is not only EMA, which assists the

developer; EMA is also assisted by the developer. This approach requires a running vehicle. Thus, it is crucial to develop an applicable, comprehensive methodology which is running and which can be the base for the further development of methodologies (prototyping for methodologies).

EMA contains no new AI knowledge, it is merely one way of applying AI, but it is a comprehensive, compact and extendable way. EMA is meant to be an extendable framework for applying AI.

In this chapter a terminology and concepts like knowledge-based application, formalism (meta-model), model, methodology, and their relationships are introduced. The problem with these concepts is, that they often have a meaning specific to the given methodology. All teams developing a methodology have the problem of defining their basic concepts specific to their methodology, because the definitions of these concepts must be harmonized. Therefore, a terminological problem appears and often the decision has to be made either giving a wellknown established term a (partly) new meaning or creating a new term. In EMA as in other approaches most often the second way has been chosen, although some established terminology has similar meaning. A negative impact of this is, that it makes these kinds of papers rather difficult to understand. Throughout this paper the EMA concepts are related to some other methodologies in order to facilitate the understanding of the definitions.

2.1. Process, Knowledge and Knowledge-Based Application

A *knowledge-based application* (KB application) is considered as a *process*. A process is defined as consisting of a *processor*, of *knowledge-information* which is driving the processor, and of *data-information* (Figure 1). Sometimes the data-information is not of interest and is then omitted. Both knowledge and data-information is information. All information which drives the processor is called knowledge-information. Information which is processed by the processor is called data-information. It is possible that some information is both knowledge and data-information.

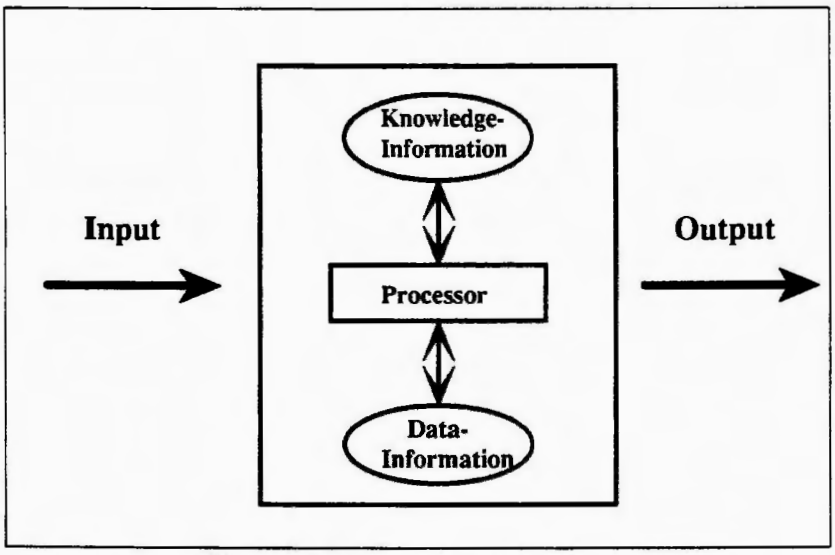


Figure 1 Components of a process

From the EMA perspective the world is seen as consisting of processes. Therefore, this concept is essential, although it is a rather trivial one. The process of EMA can be compared with the task of PROTÉGÉ (Puerta, Egar, Sampson, and Musen, 1992).

Some examples shall outline the generality of the concept process.

Example 1:
Assume a human is solving a problem guided by a book. Then the knowledge in the book is the knowledge of the process and the human is the processor.

Example 2:
Assume an application is running on a computer. Then the machine code is the knowledge and the computer is the processor.

Example 3:
Assume a typical expert system. The expert system shell plays the role of the processor. The expert's knowledge put into the shell is then the knowledge of the process.

In all three examples knowledge is modeled in a *formalism* which the processor is able to interpret. This means for the first example that other books which are written with the same formalism can also be interpreted by the human.

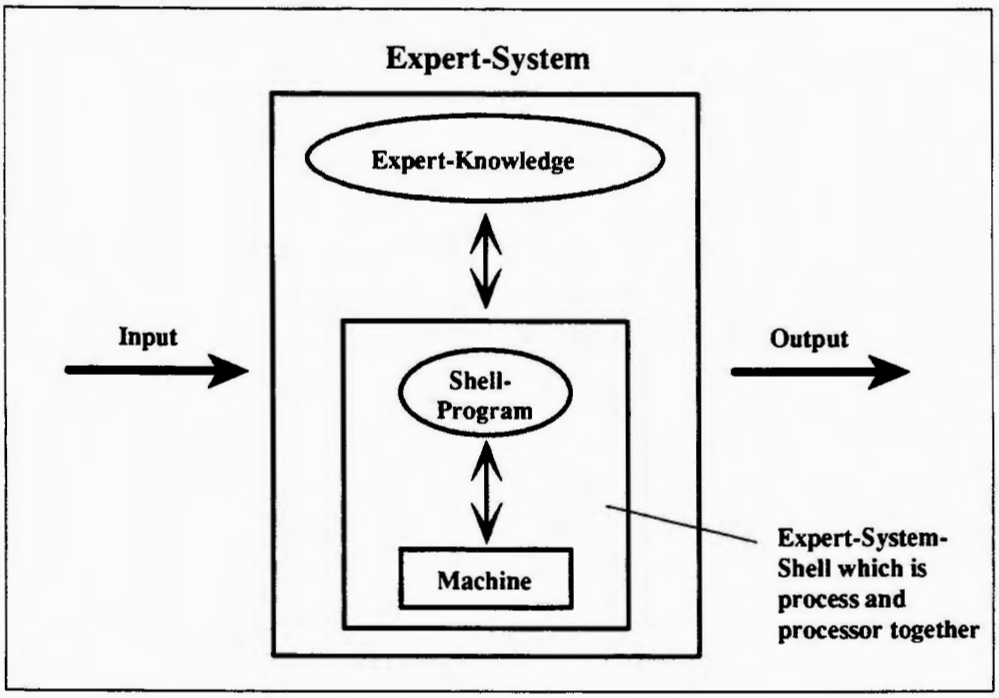


Figure 2 Processor as a process

A processor itself can be regarded again as a process. In Figure 2 the expert system shell is processor and process together. Therefore, a processor contains knowledge as well and it is arbitrary what part of the knowledge is considered as the knowledge outside the processor and which inside. But the implication is, that relative to a process the process-knowledge is

variable and the processor-knowledge invariable. It is defined for a given process which knowledge is outside the processor and which is inside.

Since a KB application is seen as a process running on a given processor, the development of a KB application means to develop the knowledge of the process, because the processor is assumed as the given, invariant and general part of the process. In order to develop the knowledge of the process, the methodology must know about the processor. The processor provides building-blocks which he is able to interpret and execute. The methodology must know about these building-blocks, in order to instantiate and configure them. The kind of provided building-blocks determine the kind of application which can be built. Furthermore, the building-blocks are specific to the methodology. Developing a methodology means first of all to define the building-blocks. A requirement to a methodology is that only the application-specific knowledge has to be developed, which means that all the general knowledge has to be provided by the methodology. The characteristic of an elementary building-block is its blackbox behaviour which makes it impossible to exercise control on an elementary building-block. This means, even if a building-block is appropriate to solve a certain problem, this building-block cannot be instantiated, if control on this problem-solving has to be exercised. This problem-solving has then to be modeled by smaller-grained building-blocks. It is the same with problem-solving techniques which have an application-specific part. Methodologies which provide building-blocks of one single and high granularity level, which is typical for role-limiting methodologies, are limited in their applicability. A way to overcome this limitation is presented in (Musen, 1992): a smaller-grained building-block is added which serves for the construction of the higher level building-blocks. This way, the high-level building-blocks become flexible. EMA (Spirgi and Wenger, 1992) and KADS (Wielinga, Schreiber, and Breuker, 1992) provide building-blocks of various granularity size. This allows the building of higher-grained building-blocks with smaller-grained ones.

2.2. The Formalism carrying Knowledge

The knowledge of the process is organized according to a certain *formalism*. Since this knowledge consists in a configuration of instantiated building-blocks, the formalism is composed of these building-blocks and their relationships. The formalism for modeling knowledge must fulfil the following requirements:

- It has to be rich enough to enable the development of the required set of applications and the required characteristic of an application (input-output characteristic).

- The interpretation by the processor must be efficient.

- It must model the problem adequately on a specification level.

- It must be robust which means that the knowledge should be usable although it may be faulty, inconsistent and incomplete.

- It must be flexible which means easy to manage, to build, to modify, to verify and to validate. A modification must only have a limited local impact and must be made on a running process (prototyping capability).

- It must allow non-redundant modeling of the knowledge.

- The formalism itself must be easy to extend.

The current computers demand knowledge in a procedural formalism to obtain high performance, but a procedural formalism contradicts the specification requirements. This means that a significant gap between a formalism with high executable power and one with high specification power exists. The solution seems to be to develop knowledge in a specification-oriented formalism and then to transform knowledge into a powerful executable formalism which is then interpreted by a processor.

The criterion of prototyping capability demands that a limited modification of the specification knowledge results in a limited modification of the executable knowledge. This implies that first the building-blocks of the specification-oriented formalism must correspond to the building-blocks of the executable formalism (structure preserving), and that second the building-blocks must be decoupled from each other. In KADS (Wielinga, Schreiber, and Breuker, 1992) this principle is called 'structure-preserving design'.

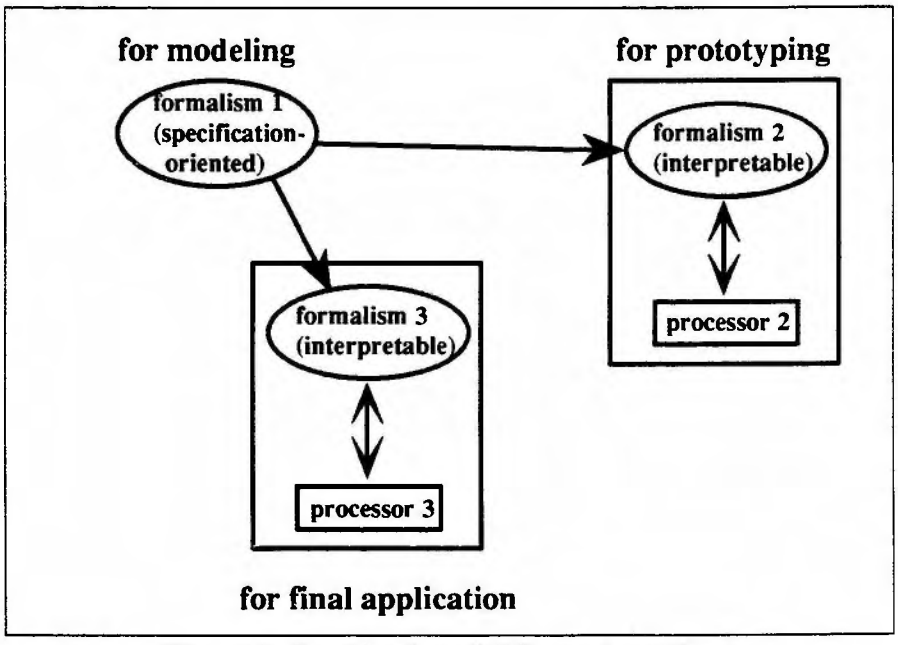


Figure 3 Combination of different formalisms

Figure 3 shows a possible combination of three different formalisms. Formalism 1 is specification-oriented, formalism 2 is similar to formalism 1 but is interpretable (for prototyping), and formalism 3 is executable with high performance. This combination is currently used in EMA. Formalism 1 is called the meta-model of the EMA-model. If a processor was built for formalism 1, then formalism 2 would be superfluous. In the case of EMA, it was more convenient to take a commercial environment for formalism 2 with processor 2 (ART-IM of Inference) and to build a transformation process (generation) between formalism 1 and formalism 2.

The criterion for robustness demands decoupled building-blocks which can be executed in parallel. This way, if one piece of knowledge cannot be executed or is executed the wrong way, other parts do their job without being affected.

The criterion for non-redundant knowledge leads to the principle of abstraction, because common characteristics can be bundled on a higher level of abstraction (e.g. object-oriented approach).

As a conclusion, the specification-oriented formalism must cover at least the principles of abstraction, parallelism and decoupling. The ideal case would be to have only one formalism which implies that the machine (processor) would be able to interpret abstraction, parallelism and decoupling efficiently which is tried to achieve in KARL (Angele, Fensel, Landes, and Studer, 1991). A step in this direction is the EMA+P (P for parallel) which will be an effort implementing abstraction and the blackboard principle on parallel computers. This step is related to work done on the field of Parallel Production Systems (Kuo and Moldovan, 1992; Kikuchi, Yukawa, Matsuzawa, and Ishikawa, 1990). Such an implementation is a processor which is capable of interpreting efficiently knowledge in a formalism which corresponds to the EMA formalism. Figure 4 relates qualitatively the specification power and the efficiency of execution of the different formalisms. Formalism 3 can be more efficiently interpreted by a processor than formalism 2, but it has a completely different structure compared to that of the model. This makes it difficult for prototyping.

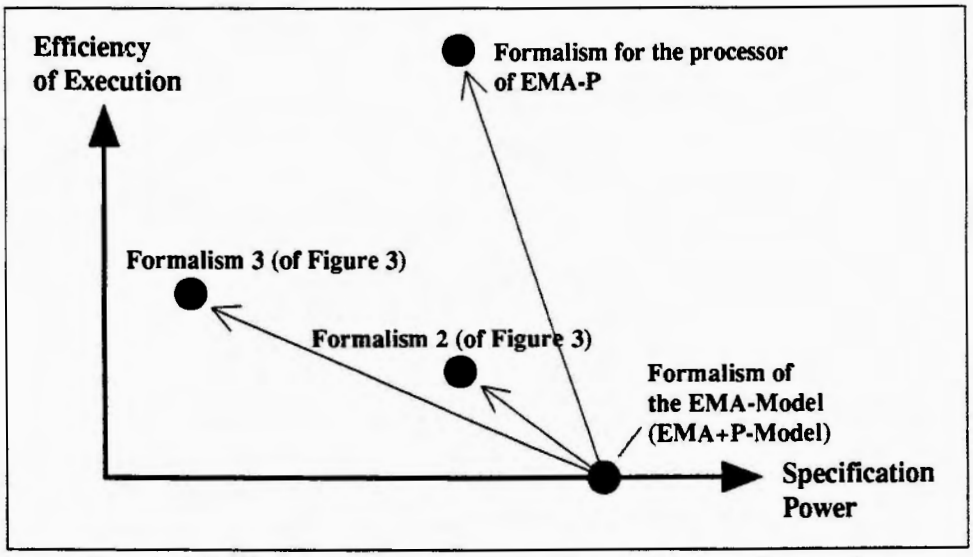


Figure 4 Different formalism compared to efficiency of execution and specification power

2.3. Methodology and Development Cycle

Application development is a cooperative task of a developer and a machine: the developer specifies assisted by EMA, and EMA develops the application according to the specifications. This task can therefore be represented by a human-based application development process (developer) and a computer-based one.

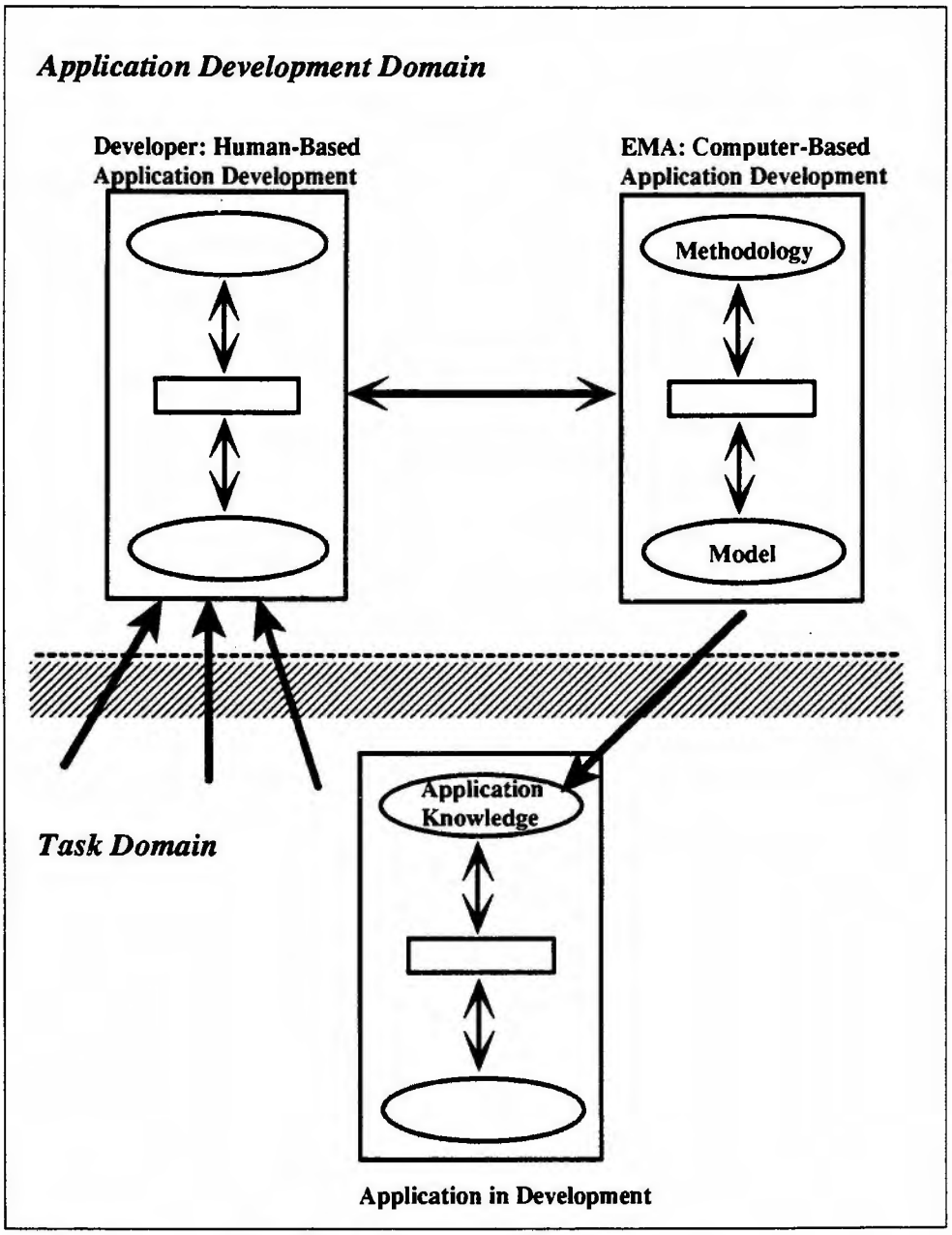


Figure 5 Application Development Cycle

Figure 5 shows the application development cycle. The input for the human-based process comes from the task domain. A strong interactivity (input and output) between the human-based and computer-based development process exists. The methodology is seen as the knowledge-information and as the model the data-information of the computer-based development process. The output of the computer-based development process is the knowledge-information of the application process. At present, only the developer is capable of acquiring information of the task domain which the developed application is part of. The difference and contradiction between the current characteristic of the application and the requirements (which are part of the specification) are the driving forces for knowledge acquisition (verification and validation). The methodology knowledge drives the processor

(of the EMA process) for developing the model. The developer is assisted by EMA in order to provide the specification.

2.4. The Model

The information contained in the model and the knowledge-information of the application are related to each other, because the latter is generated from the former. The question is whether principal differences exist between these two formalisms. The principal in EMA to specify the application until the specification is complete demands for a formalism adequate for specifying. Specification takes place on different levels, e.g. specifying a problem-solving technique is on a different level than specifying a function, but both is specification. Therefore, no distinction between different levels, such as epistemological, conceptual and executable level is made in EMA. Specification takes simply place with a set of building-blocks of various granularity size. Thus, also a programming language, such as Prolog or C and also the formalism of the application's knowledge information, are inadequate formalisms for specification, because they do not fulfil the requirements of chapter 2.2.

Three levels carrying knowledge can be identified: the *specification level* which is the data-information of the EMA process and which is called the model; the *execution level* which is the knowledge-information of the application process; the *processor level* which is the processor for the applications. An analogy, but just an analogy, can be drawn to Newell's knowledge and symbol level (Newell, 1982) - the specification level corresponds to the knowledge level, the processor level corresponds to the symbolic level. As outlined in chapter 2.2., the execution level is not a principal level. It is a level which is convenient for the present conception, but it would be superfluous, if the specification could be directly interpreted by the processor.

These three levels contain different types of knowledge. The processor level carries the knowledge of the building-blocks. This knowledge is general and invariant from the developer's perspective (see chapter 2.1). It is knowledge which is part of the methodology-specific knowledge. The execution level contains knowledge generated from the specification level. In addition, this level contains knowledge which is necessary to bridge the gap between the building-blocks of the specification level and those of the processor (generation knowledge). Assume as an example, the processor knows how to interpret building-blocks of the language C, then generation knowledge is to know how to transform a specification based on object-oriented building-blocks into building-blocks of C, such as pointers and arrays. This generation knowledge is methodology-specific and general for all the applications. The most interesting level is the specification level. This level contains application-specific knowledge, but it also contains specifications of general or domain-specific problem-solving techniques.

3. THE APPLICATION EMA

3.1. The Components of EMA

The application EMA (process EMA) has to perform the requirements and characteristics outlined in chapter 2. EMA consists of several subsystems (components of EMA) around its central part - the model (Figure 6).

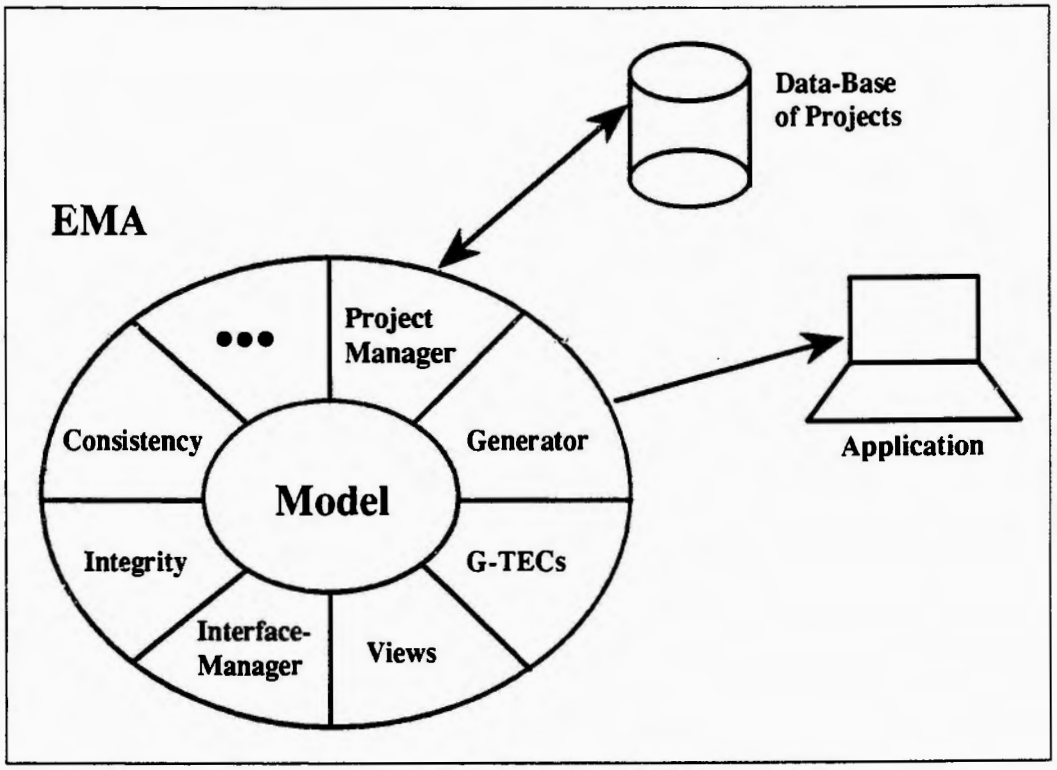


Figure 6 Subsystems of EMA

The model stores the information of the application which is in development, structured according to a formalism. The subsystems serve the single goal of processing and managing the information of the model. The formalism is responsible for the kind of information contained in the model, and therefore, determines the basic characteristic of a methodology. Developing a good methodology means above all finding a good formalism.

3.2. The Model and Meta-Model

In EMA the formalism is called the *meta-model*. The building-blocks and their relations define the meta-model of the model. They are used for modeling knowledge and must therefore possess high modeling power. Modeling the formalism means building the meta-model of the model. The building-blocks are therefore also called meta-concepts, because they are the concepts of the meta-model. A formalism is knowledge as well and it is therefore logical to model it with its own formalism. This means that the meta-model and the model are built with the same building-blocks. The modeling in EMA is presented in the related paper (Spirgi and Wenger, 1992).

3.3. The Views

The model in EMA is a structured amount of information. Information is something abstract and therefore the model is abstract, too. The information within the model can be seen from different aspects. Such an aspect is called a view and is a representation. The views make the model visible and are the interface for the developer. A view can also be called representation. The model is originary and compact although all different kinds of information can be carried in it. This way, the information is not redundant. If any new kind of information needs to be stored in the model, the model can be extended. There is absolutely no reason for having different models in a methodology. It is even a drawback, because redundancy is introduced which causes drawbacks in respect to the simplicity, applicability and manageability of a methodology.

Any number of views can be imagined, because a model can be seen from many different aspects. A view does not exist for its own sake: it is a medium for communicating with the model and is therefore a developer interface. The views are built as graphical editors. A view consists of a set of representable, graphical symbols with given meanings. More about the views has been presented in (Spirgi and Wenger, 1992).

3.4. The G-TEC Manager

The G-TEC (Generic-TECnique) approach is outlined in (Spirgi, Probst, and Wenger, 1991) and has some parallels with generic task (Bylander and Chandrasekaran, 1987; Chandrasekaran, 1986), the interpretation model approach of KADS (Breuker et. al., 1987) and role-limiting approaches, such as PROTÉGÉ (Puerta, Egar, Sampson, and Musen, 1991; 1992) and SBF (McDermott et. al., 1990; Marques et. al.,1991; 1992). A G-TEC can be seen as a generic process (generic application) which solves a certain kind of problem. G-TECs are high level problem-solving building-blocks. We distinguish between CG-TECs (Conceptual G-TECs) and TG-TECs (Technical G-TECs). G-TECs are modeled on the specification level which makes it possible to exercise control on them. Furthermore, they can be used both as building-blocks which can be instantiated and as templates for new building-blocks.

3.4.1. Conceptual G-TECs

A CG-TEC solves a certain problem. In EMA solving a problem means to provide a certain information (e.g. a planning, an assignment, an evaluation information). A CG-TEC is related to exactly one kind of problem and is herewith problem-specific. The classification of the information provided is used to classify the CG-TECs. The classifying question is: What is the provided information used for and interpreted as by the user? Answers are: for planning, for an assignment, for an evaluation, etc.

CG-TECs are an important and powerful help for knowledge acquisition, because they are problem-solving patterns which can be looked for in the task domain. The first step is to model a certain required information. This information is classified according to the classification of the CG-TECs. If it is possible to classify the required information, then one or more CG-TEC(s) are available. The model of an available CG-TEC serves then as a pattern for acquiring the application-specific concepts of the task domain. An example is

outlined in (Spirgi and Wenger, 1992), where CG-TECs are applied for solving an assignment problem. An example is the CG-TEC CONF-CLASS which has the concept 'resource'. The developer has to look for a concept in the task-domain which can play the role of the concept 'resource' (the concept becomes a meta-concept for the concept found). If the developer has all the necessary information for a CG-TEC, then this CG-TEC can be instantiated, integrated in the model and applied.

3.4.2. Technical G-TECs

The TG-TECs are independent in respect to a specific problem. They are algorithms which can be applied in different areas of problem-solving. The information they provide cannot be classified by answering the question: What is the provided information used for and interpreted as by the user? Search techniques e.g. are seen as TG-TECs, because they perform a certain mechanism where the processed and provided information is neutral to problems. Therefore, a TG-TEC cannot be classified according to its produced information. It is classified according to its mechanism.

3.4.3. Positioning

A technical but essential difference to PROTÉGÉ, SBF and the Generic Task approach is the modeling of problem-solving. The complete specification of a G-TEC is stored in the model. This means that an executable G-TEC is generated by the generator solely from the specification of the model and is herewith not a blackbox from the methodology perspective. Another important difference is the position of generic problem-solving. In EMA, the G-TECs are predefined models which are at the developer's disposal and which can be included in the application model, but absolutely no difference exists between parts of the application model which stem of the G-TEC library and parts which have been specified by the developer by means of the views. This means, the G-TECs are an important and powerful subsystem of EMA, but it is possible to develop an application without any G-TEC at all. This kind of model-based problem-solving approach has the advantage that a G-TECs can be instantiated, but that a G-TEC can also be taken as an original (template) and then adapted, modified and extended. This corresponds strongly to the PROTÉGÉ-II (Puerta, Egar, Sampson, and Musen, 1992) approach which allows also that a problem-solving method can be adapted to the domain or the task. It also corresponds to the principle of 'knowledge differentiation' of KADS (Wielinga, Schreiber, Breuker, 1992).

3.5. The Interface-Manager

The application interacts with its environment. The environment is partitioned in different areas, such as the data-base, the screen and the printout area. These areas are also modeled by EMA. The screen and the printout area are specific to the application. The data-base area can be shared with more than one application. These areas are outside the process (Figure 1). The modeling of the areas enables us to identify the information accessed and manipulated by the process. The knowledge within the process can be extended to drive the *accesses* and manipulations (in EMA called *effects*) of the areas. Thus, the knowledge of the interaction with the environment is modeled in EMA on the specification level.

The interface manager consists of the managers of the different areas such as the *screen-manager*, the *printout-manager* and the *data-base-manager*.

3.6. The Integrity- and Consistency-Manager

The integrity-manager is necessary for a kind of housekeeping. So for example objects are not directly deleted, they are marked for deletion. By turning on the integrity-manager, different kinds of housekeeping are performed.

The consistency-manager is based on the relationships and restrictions of the meta-model. For instance two objects are not allowed to be related by an isa-relation and association at the same time. The consistency-manager allows inconsistencies, but it generates a message which exists as long as the inconsistency exists.

3.7. The Project-Manager

Managing projects is saving and restoring projects to/from the data-base and it seems to be a rather simple task. But the managing of G-TECs, the dynamic loading of G-TECs and the managing of the whole amount of information about an application is a challenging task.

3.8. The Generator

The generator is necessary, because the formalism of the model is not directly interpretable. The generator runs in the full and incremental generator mode. The incremental mode is required, because the EMA methodology-application and the developed application run in parallel. This means, a modification in the model is immediately transferred to the application. This principle is denoted as *model-based prototyping*.

4. DEVELOPING A METHODOLOGY

In EMA, developing a methodology means developing a methodology-application, because the methodology is the knowledge-information of a process (Figure 5). Figure 7 shows the different domains: methodology development domain, application development domain and application task domain. The methodology is developed in the methodology development domain. The developing methodology is EMA itself, the task domain is the domain of application development. The developer of EMA acquires information of this task domain. In the methodology development domain the model in EMA contains the meta-model of EMA. The methodology EMA which is the knowledge-information of the process EMA is generated from this model . In the application development domain EMA is used for developing applications. In this domain the developer acquires information of the application task domain. Thus, developing a methodology is the same as developing an application. Moreover, it is a criterion for a general purpose methodology that it can be developed with the help of itself.

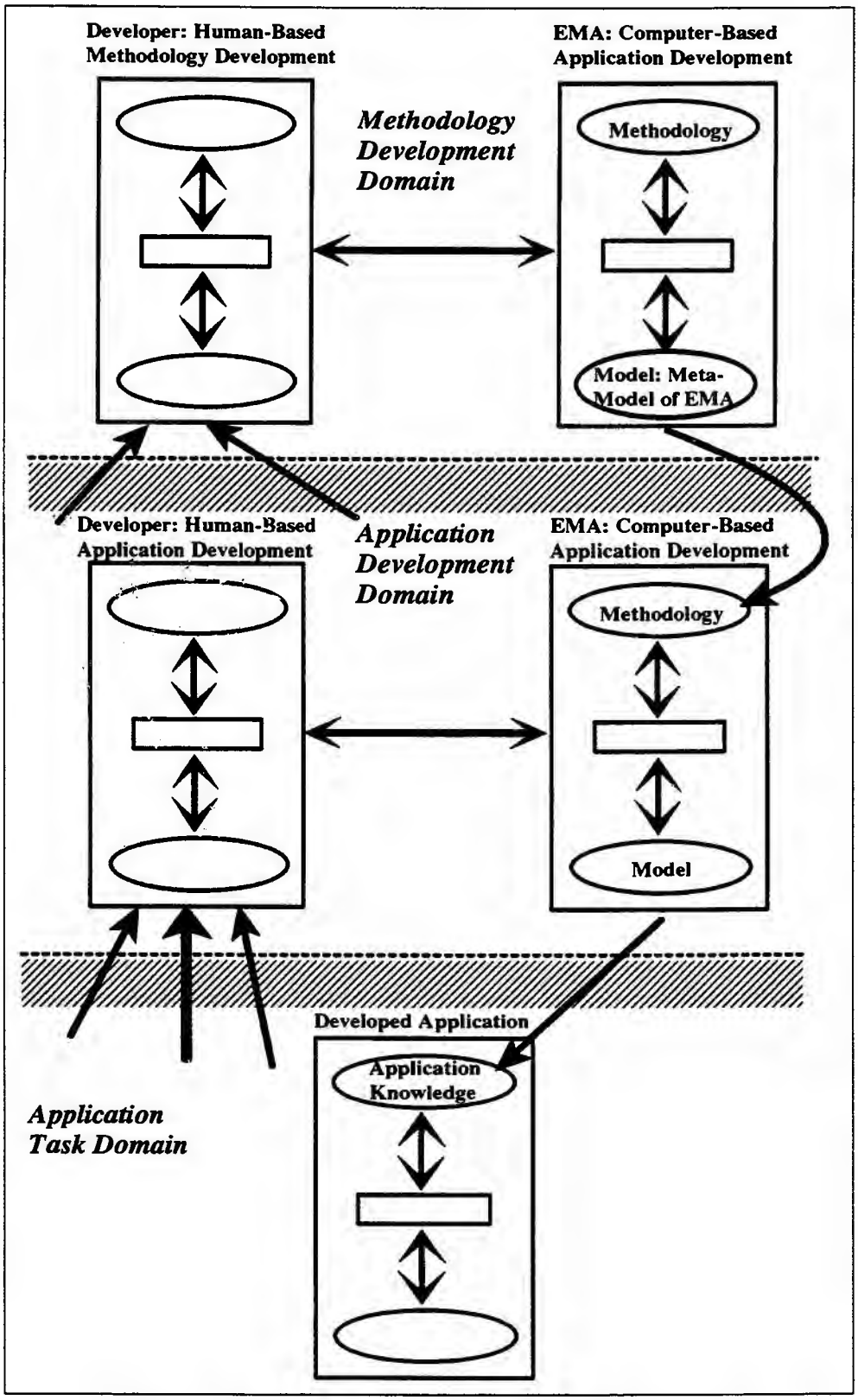


Figure 7 Methodology and Application Development Cycle

5. RELATING EMA TO OTHER APPROACHES

EMA has been developed as a generalisation of experience resulting from the development of knowledge-based applications (Wolf, Wenger, and Kirchmayr, 1991; Hottiger and Wenger, 1992). Thus, EMA has been permanently streamlined by the development of applications. When looking at other approaches, parallels and differences have appeared. Especially differences between other approaches and EMA have inspired the development of EMA. Among others the PROTÉGÉ (Puerta, Egar, Sampson, and Musen, 1991; 1992), the KADS (Wielinga, Schreiber, and Breuker, 1992), the SBF (McDermott, Dallemagne, Klinker, Marques, and Tung, 1990; Marques, Klinker, Dallemagne, Gautier, McDermott, and Tung, 1991; Marques, Dallemagne, Klinker, McDermott, and Tung, 1992) approach and the KSSn family (Gaines, 1990) should be mentioned. A comparison has been elaborated to KADS, SBF and PROTÉGÉ.

5.1. A General Representation for the different Approaches

Before starting to relate various methodologies with EMA, a general representation for methodologies is presented (Figure 8). Various tasks have to be more or less carried out by the developers depending on the methodology. This general representation is represented in a kind of information-flow diagram. A first task is to develop generic models which is normally done by the developers of the methodology. A task carried out during application development is to match and to modify generic models. Next task is to instantiate the generic models and to configure the instantiations. An additional way is to build the model from scratch by instantiating building-blocks and to configure the instantiations. One could say that a generic model is also a building-block, but it is a building-block composed of elementary, non-divisable building-blocks. The implication is that the elementary building-block is invariant and a blackbox for the developer, whereas a generic model is explicitely and completely specified. In EMA terminology, the knowledge of an elementary building-block is on the processor level, the knowledge of a generic model on the specification level.

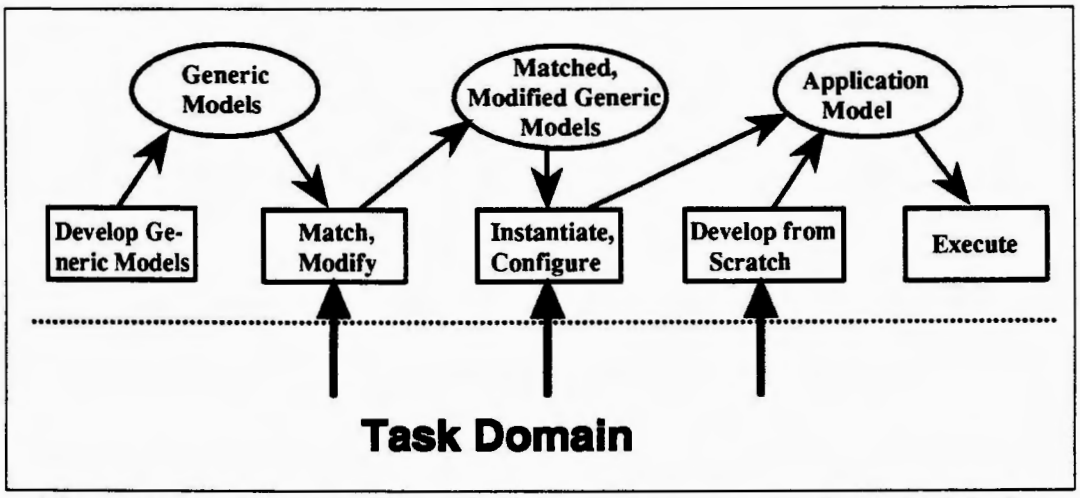


Figure 8 Information-flow diagram of a general representation for methodologies

5.2. Comparison to KADS

On this highly abstract representation (Figure 8) EMA and KADS are similar. All the tasks described are carried out in KADS and in EMA as well. KADS and EMA are both methodologies thought to be applied by professional developers and not by domain experts or end-users. But, when going into details, many differences exist.

Multiple-model approach:
A first difference is the multiple-model approach of KADS which stands in opposition to the single-model approach of EMA. This difference has its origin in the different meaning of 'model'. The models in KADS are representations which correspond to the views in EMA. The model of EMA contains the information about the different views in a non-redundant way.

Organisational model:
The organisational model of KADS has its equivalent in the representation of processes. Figure 5 is an example. Moreover, Figure 5 emphasizes that the application is part of the task domain and so of the organisation.

Task model:
Instead of the task model and its decomposition, EMA stores requirements for the application. The requirements are expressed as required information the application has to provide. Instead of task decomposition in EMA, the required information is decomposed. This has an important advantage, because tasks are fuzzy concepts, whereas information is exactly specified. Therefore, decomposition in KADS is fuzzy. This difference has its origin in the principle of EMA that the application has to provide the required results, whereas how the results are produced is secondary. In EMA the information is in general the primary focus of the modeling and not the processing of information.

Model of cooperation:
The model of cooperation is used in KADS for modeling user-interaction. In EMA user-interaction is tackled very differently. The exchanged information plays the key role. This information is part of the information areas of which conceptual models exist. Interaction between an application and these areas is done by activation, accesses and effects (Spirgi and Wenger, 1992).

Model of expertise:
The model of expertise in KADS is based on Newell's (Newell, 1982) distinction between knowledge and symbol level. In KADS the knowledge level is taken for the model of expertise. In EMA the specification level is the counterpart to the knowledge level and the processor level that to the symbol level. The term knowledge level cannot be applied in EMA, since knowledge has in EMA its own specific definition (see chapter 2.1.). In EMA, whether some information is knowledge or not is not a question of the form (e.g. declarative, computational, etc.); if information can be directly or indirectly (information has first to be transformed into another form) used by a processor for executing a task, then this information is considered as knowledge. Thus, information in a computational form can be knowledge as well.

The layers:
The different layers in KADS correspond to the different views in EMA. The concepts of the domain knowledge in KADS have their equivalent meta-concepts in EMA.

Inference knowledge:
The inference knowledge has its analogy in the agent view of EMA (Spirgi and Wenger, 1992).

Task knowledge:
The task knowledge corresponds to the information in the processing part of an agent. In EMA this knowledge is specified in more detail in order to generate executable knowledge. A task corresponds in EMA to the processing part of an agent, a subtask to an application-specific function and a knowledge source to a general function (Spirgi and Wenger, 1992).

Strategic knowledge:
Parts of the knowledge in the activation part of an agent can be seen as strategic knowledge.

Knowledge source:
Reusability in KADS and EMA is rather similar. Knowledge source in KADS has as corresponding building-block in EMA general function (Spirgi and Wenger, 1992) and TG-TEC (Spirgi, Probst, and Wenger, 1991) . The problem of finding a typology of knowledge sources corresponds in EMA to that of finding a typology for general functions and TG-TECs.

Interpretation model:
The interpretation model has a rather similar building-block in EMA called CG-TEC which consists of a model on the specification level.

Conceptual and design model:
KADS separates the conceptual and the design model, whereas in EMA one model exists. A main difference between KADS and EMA lies in the completeness of the specification. In KADS the specification within the conceptual model is not complete which means that the conceptual models do not contain all information necessary for the implementation of an application. In EMA the model contains all the information for generating the application and is therefore complete. Developing the design model from the conceptual model is a major and still magic step in KADS.

Formal language:
A formal language developed for the KADS models of expertise, such as KARL (Angele, Fensel, Landes, and Studer, 1991), $(ML)^2$ (van Hermelen and Balder, 1992) and OMOS (Linster, 1992), can be seen as a meta-model and corresponds to the meta-model of EMA or at least plays the same role. The difference is that the meta-model in EMA is modeled with its own meta-concepts.

One of the most serious problems of KADS is its shortcoming to support the building of a running system. The building-blocks of KADS are rather informal than formal. Therefore, languages (formalisms) have been developed to overcome this deficiency, such as KARL, $(ML)^2$ and OMOS. In particular the objectives of KARL are fairly similar to those of EMA.

The main objectives of KARL are: to allow a formal specification of the model of expertise (specification level of EMA), to map the specification automatically to an executable formalism, and to provide graphical representations (views of EMA). Another deficiency is KADS's heaviness in order to be applied. An attempt to improve this is the KEEP Model (Neubert and Studer, 1992) which describes the necessary knowledge engineering activities which have to be carried out.

5.3. Comparison to Spark, Burn and Firefighter (SBF)

SBF has its emphasis on supporting the generic model approach. No development from scratch is supported. SBF is modeling a task of the task domain by its task model. This stands in contrast to EMA, because in EMA the task domain in not modeled. A configuration of mechanisms (or task method) corresponds to the model of a CG-TEC on the specification level. A mechanism has its EMA counterpart in the concept agent.

The most important difference to EMA is that the developer is rather seen as an end-user or a domain expert than as a professional developer. This has many implications: no developing from scratch is supported which obviously also makes no sense for end-users without programing skills; the mechanisms are blackboxes for the developer (the internals of a mechanism are hidden to the developer); emphasis and key area of research is on the matching of the task with a task model and the mapping to a computational model; the goal of the methodology is not (and cannot be) at the moment to develop highly sophisticated applications, but to develop useful end-user applications. EMA in contrast addresses itself to the professional developer in order to help him to develop in particular highly sophisticated solutions.

5.4. Comparison to PROTÉGÉ

Similar to SBF, PROTÉGÉ has its emphasis on the generic model approach and does not support development from scratch. It also addresses itself to non-professional developers - especially to domain-experts. This leads to the same differences to EMA as mentioned already in the comparison to SBF.

In PROTÉGÉ the concept method corresponds to a CG-TEC in EMA, a mechanism to a TG-TEC. A main focus is the method assembly which is in EMA not supported as much at the moment. PROTÉGÉ seems to be a compact and a sufficiently flexible approach for developing also highly sophisticated solutions.

6. DISCUSSION AND PERSPECTIVES

EMA is intended to be a framework for all kinds of new problem-solving techniques. By integrating new approaches, such as case-based reasoning, new approaches are combined with approved techniques. This way, new techniques get their position and can be reasonably applied, because often expectations are too high for new approaches. A first major short-term effort is to make EMA available for other groups in SBC (Swiss Bank Corporation) for wider use. A second permanent effort is to extend and improve EMA. A third long-term effort is to develop a parallel computing environment in order to build parallel knowledge-based

applications. A fourth effort is to establish EMA as a knowledge-based environment for commercial software development.

REFERENCES

Angele Jürgen, Fensel Dieter, Landes Dieter, Studer Rudi (1991). KARL: An Executable Language for the Conceptual Model, Proceedings of the 6th AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, KAW-91 Banff, pp 1-1-1-20

Breuker J. (ed), Wielinga B., Schreiber G., de Greef P., de Hoog R., van Someren M., Wielemaker J., Billault J.P., Davoodi M., Hayward S. (1987). Model Driven Knowledge Acquisition: Interpretation Models, Del A1, Esprit p1098, University of Amsterdam

Bylander T., Chandrasekaran B. (1987). Generic tasks for knowledge-based reasoning: the "right" level of abstraction for knowledge acquisition, Int. J. Man-Machine Studies (1987) 26, pp. 231-243

Chandrasekaran B. (1986). Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design, IEEE Expert (Fall 1986), pp 23-30

Gaines B. R., (1990). Knowledge representation servers: a generic technology for knowledge acquisition and knowledge-based systems, Proceedings of the First Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop, JKAW'90, pp 413-430

Hottiger S., Wenger D. (1992). MOCCA: A set of instruments to support mortgage credit granting, Proceedings of the IAAI-92 Conference, pp 135-152

Kikuchi J., Yukawa, T., Matsuzawa k., Ishikawa T. (1990), Presto: A Bus-Connected Mulitprocessor for a Rete-Based Production System, CONPAR 90-VAPP IV, Lecture Notes in Computer Science, Springer-Verlag, pp 63-74

Kuo S., Moldovan D. (1992), The State of the Art in Parallel Production Systems, Academic Press Inc., pp 1-26

Linster Marc (1992). Linking Modeling to Make Sense and Modeling to Implement Systems in an Operational Modeling Environment, in Proceedings of the 6th European Knowledge Acquisition for Knowledge-Based Systems Workshop (EKAW-92), Lecture Notes in Artificial Intelligence 599, Current Developments in Knowledge-Acquisition-EKAW'92, Springer- Verlag, pp 55-74

Marques D., Klinker G., Dallemagne G., Gautier P., McDermott J., and Tung D. (1991). More Data on Usable and Reusable Programming Constructs, Proceedings of the 6th AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, KAW-91 Banff, pp 14-1-14-19

Marques D., Klinker G., Dallemagne G., Gautier P., McDermott J., and Tung D. (1992). Easy Programming, IEEE Expert, June 1992, pp 16-29

McDermott J., Dallemagne G., Klinker G., Marques D., Tung D. (1990). Explorations in How to Make Application Programming Easier, Proceedings of the First Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop, JKAW'90, pp 134-147

Musen Marc (1992). Editorial - Overcoming the limitations of role-limiting methods, Knowledge Acquisition, Vol. 4 Nr. 2, Academic Press Limited, pp. 165-170

Neubert Susanne, Studer Rudi (1992). The KEEP Model, a Knowledge Engineering Process Model, in Proceedings of the 6th European Knowledge Acquisition for Knowledge-Based Systems Workshop (EKAW-92), Lecture Notes in Artificial Intelligence 599, Current Developments in Knowledge-Acquisition-EKAW'92, Springer- Verlag, pp 230-249

Newell A.(1982). The knowledge level, Artificial Intelligence, pp. 87-127

Puerta A., Egar J., Sampson T., Musen M. (1991). A multiple-method knowledge-acquisition shell for the automatic generation of knowledge-acquisition tools, Proceedings of the 6th AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, KAW-91 Banff, pp 20-1-20-19

Puerta A., Egar J., Sampson T., Musen M. (1992). A multiple-method knowledge-acquisition shell for the automatic generation of knowledge-acquisition tools, Knowledge Acquisition, Vol. 4 Nr. 2, Academic Press Limited, pp. 171-196

Spirgi S., Probst A., Wenger D. (1990). Knowledge Acquisition in a methodology for knowledge-based applications, Proceedings of the First Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop, JKAW'90, pp. 382-397

Spirgi S., Wenger D., Probst A. (1991). Generic Techniques in EMA: A Model-Based Approach for Knowledge Acquisition, Proceedings of the 6th AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, KAW-91 Banff, pp 31-1-31-13

Spirgi S., Wenger D. (1992). Project Sisyphus: EMA approach Version 2, Arbeitspapiere der GMD 630, March 92, Sisyphus '92: Models of problem-solving, EKAW'92

Spirgi S., Wenger D. (1992). Modeling in EMA, Proceeding of the 7th AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, KAW'92 Banff

van Hermelen F. and Balder J. (1992). $(ML)^2$: A formal language for KADS models of expertise, Knowledge Acquisition, Vol. 4 Nr. 1, Academic Press Limited, pp. 127-161

Wielinga B.J., Schreiber A.Th. and Breuker J.A. (1992). KADS: a modelling approach to knowledge engineering, Knowledge Acquisition, Vol. 4 Nr. 1, Academic Press Limited, pp. 5-53

Wolf M. F., Wenger D., Kirchmayr K. (1991). CUBUS: An assistant for fundamental corporate analysis, Proceedings of the IAAI-91 Conference, pp. 271-291

A STUDY ON TASK DESCRIPTIONS USEFULNESS FOR TASK KNOWLEDGE ACQUISITION

Yuri A. Tijerino* **David Marques****
Riichiro Mizoguchi* **Georg Klinker****
Tadahiro Kitahashi* **John McDermott****

*I.S.I.R. (SANKEN), Osaka University. 8-1 Mihogaoka, Ibaraki, Osaka 562 Japan
**AIRG, Digital Equipment Corporation. 111 Locke Drive, Marlboro, MA, 01752 USA

Abstract

Much is yet to be expected from knowledge acquisition programs because they are not yet able to "talk" with task performers in an intelligent enough manner. In other words most knowledge acquisition programs do not pass the Turing test of intelligence. For this reason we have developed a manual method for knowledge acquisition that couples with knowledge acqusition programs to fill this intelligence gap. In this paper we present a taxonomy of questions which can be asked to people performing tasks to make available information needed by software and knowledge engineers to generate application programs connected though workflow managers that fit into workplace rather than intrude on it.

1. Introduction

In this paper, we present a manual method for knowledge acquisition that couples with knowledge acquisition tools for effective generation of application programs useful in cooperative tasks. This method focuses on task performers interactions with other performers by exploring those interactions with a number of questions aimed at clarifying what is done throughout those interactions. Our main hypotheses is that this questions can help generate answers that will be useful for knowledge engineers or software engineers to effectively and efficiently generate software artifacts that fit into workplace rather than intrude on it. Software artifacts generated fit into workplace because the questions that help generate them produce answers that make clear how the task performers trully perform their task. This answers are validated or further explored with other questions ensuring their validity and completeness. This questions where generated from an analysis performed on the need for task information that a framework being developed by the Artificial Intelligence Research Group at Digital Equipment Corporation (DEC) (McDermott et al., 1990; Klinker et al., 1991; Marques et al., 1991; Dallemagne et al., 1992; and Marques et al., 1992) currently has.

2. Background

The authors belong to two different groups, one from Osaka University and another from DEC, that have been working on separate efforts to develop knowledge acquisition tools that interactively work directly with knowledge possessors to help describe their task and generate software artifacts to aid in performing task performers (McDermott et al., 1990; Klinker et al., 1991; Marques et al., 1991; Dallemagne et al., 1992; and Marques et al., 1992). These efforts started in a comparatively similar way (McDermott et al., 1990; Tijerino et al., 1990) and evolved in two related, but different directions. The effort at the group from DEC is now focused on developing tools useful to interactively help non-programmers create useful application programs, not necessarily knowledge-based systems (Marques et al., 1992), while the Osaka University group focused on constructing tools to help domain experts interactively generate exclusively knowledge-based systems that help them in performing their task (Tijerino et al., 1992 and

Mizoguchi et al., 1992). Both of these efforts draw upon the hypothesis that it is possible to abstract reusable software artifacts that can be mapped to task descriptions, provided that such task description is generated through some task analysis process. We joined efforts because of the similarities in our goals and discovered that there was a need to generate accurate task descriptions. Although DEC's group had been using a technique[1] to extract such a task description it didn't take into consideration the information needed by software artifacts. Therefore, we designed UnitedWorlds, a task analysis methodology for cooperative tasks with the need for that information in mind.

3. An Study of a Real World Task

The following is a description of the study undertaken of a real world task. The study was done during a period of four months during which time different approaches where tested.

3.1. The task

The task, which we will call Management of Business Information for Knowledge Consulting Co., consisted of managing information about customers and potential customers of a group that sold knowledge-based solutions from Knowledge Consulting Co. The main motivation behind the study was that "business information was falling through the cracks" as John Redford and Fred Lambordini, the two main managers of the group explained.

The task at first seemed very fuzzy and some of the performers even said that it couldn't be described because they thought that it changed constantly. Other performers were very excited about the study because it promised them to understand better what others were doing and because of the prospect that it would result in application programs that would help them perform their corresponding task more effectively.

Most members of the group received information from customers and had to relay it to the corresponding member responsible of handling that information. The group was divided in various subgroups that had a whole range of responsibilities including marketing of the solutions, group operations financing, business development, artificial intelligence solutions, imaging solutions, technology transfer, training solutions and a wide range of internal and external product-based solutions. In performing the management of business information, everyone in the group thought that they knew what every one else was supposed to do. However, in reality this was very far from the truth because as John and Fred had said information was falling through the cracks and something had to be done quickly. By that they meant such things as that sometimes customers needs were not addressed properly and/or their billing were either inadequate or not timely.

This task was a challenge to describe because, though there was a description of what the group was supposed to do, most members had a different interpretation of what that description really meant. Another factor that made this an interesting task to describe was that the group was newly organized and most of its members were new comers from other groups within the company.

3.2. The task performers

Figure 1 shows the organizational chart of the members of Knowledge Consulting Co. However, the fact that these were the members of the task didn't mean that they were the task performers. So though the organizational chart gave an starting point, part of the description of the task included identification of the task performers. This was accomplished via spotlights as explained in the following sections.

---

[1] Digital Equipment Corporation offers a workplace analysis consulting service called BAM I which recently evolved into BAM II. It uses a workplace analysis technique to collectively interview members of communities of pratice peforming some task in a given workplace.

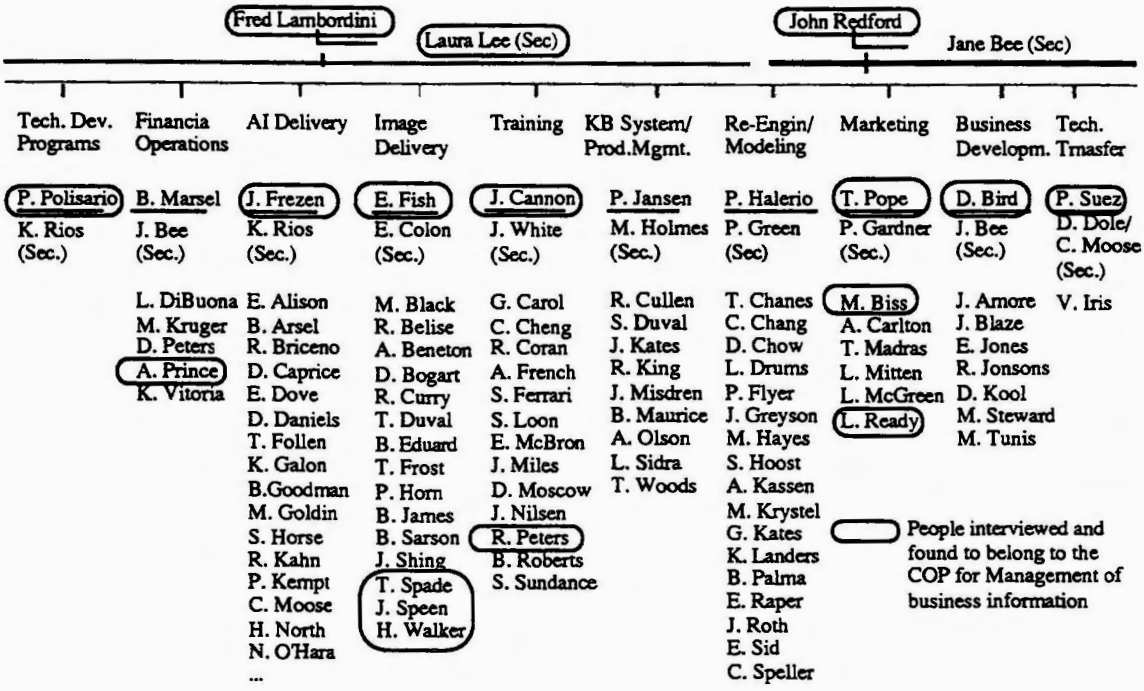


Figure 1. The organizational chart of the Knowledge Conculting Co.

3.3. The process of the study

First of all, we got John and Fred tell us what was the task that had to be described. Then, a set of about 8 persons were determined to be the members of the community of practice who were mostly in managerial positions. With that in mind we started interviewing those members with the following questions which we thought would give as some insights to describe the task.

Q1. When performing tasks related to management of business information, who do you interact with?

Q2. For each person you interact with:

- Q2.1. What is the purpose of the interaction?
- Q2.2. How do you rate importance of the interactions?
- Q2.3. Is it a formal or casual interaction?
- Q2.4. What is the frequency of the interaction?

Q3. What functions do you perform in management of business information or that have to do with management of business information? Explain.

Q4. For each of the functions you perform:

- Q4.1. Of the people mentioned above in Q1, who do you interact with in the context of each function.
- Q4.2. What do you obtain from each of them?
- Q4.3. What do you provide to each of them?
- Q4.4. What is the medium of the information?

  Face to face, phone, e-mail, forms, etc.
- Q4.5. How do you visualize or would like to visualize the data you obtain?

Q5. What kind of computer assistance do you think will help you in performing your task more efficiently?

All interviews were taped and later listened to. The reason for taping interviews is that the interviewee might say things of importance that may be overheard. So by listening to the tapes we saved a lot of time of the task performers to find answers to questions that would arise later on. Listening to the tape, we could then take two approaches: Transcribe the tape or as we listened to the tape we created an activity networks with the help of Flint, a graphical tool used to input task descriptions available as a subset of Spark.

Then, we merged the network with other networks of persons mentioned that were already interviewed. We kept a list of persons mentioned in all interviews and updated it as persons were mentioned and their relevance to the task could be determined. We then tried to interview as many relevant persons in the list as possible. We also reinterviewed as necessary. We obtained a list of contexts for each of the interviews and matched them against it to identify functions for which united views can be created including other community of practice members.

a) Flint: a tool to describe workplaces.

Steel: a tool to describe mechanisms

Smoke: a tool to associate mechanisms with workplace descriptions

Flame: a tool to describe and integrate a mechanism's data with the current workflow application

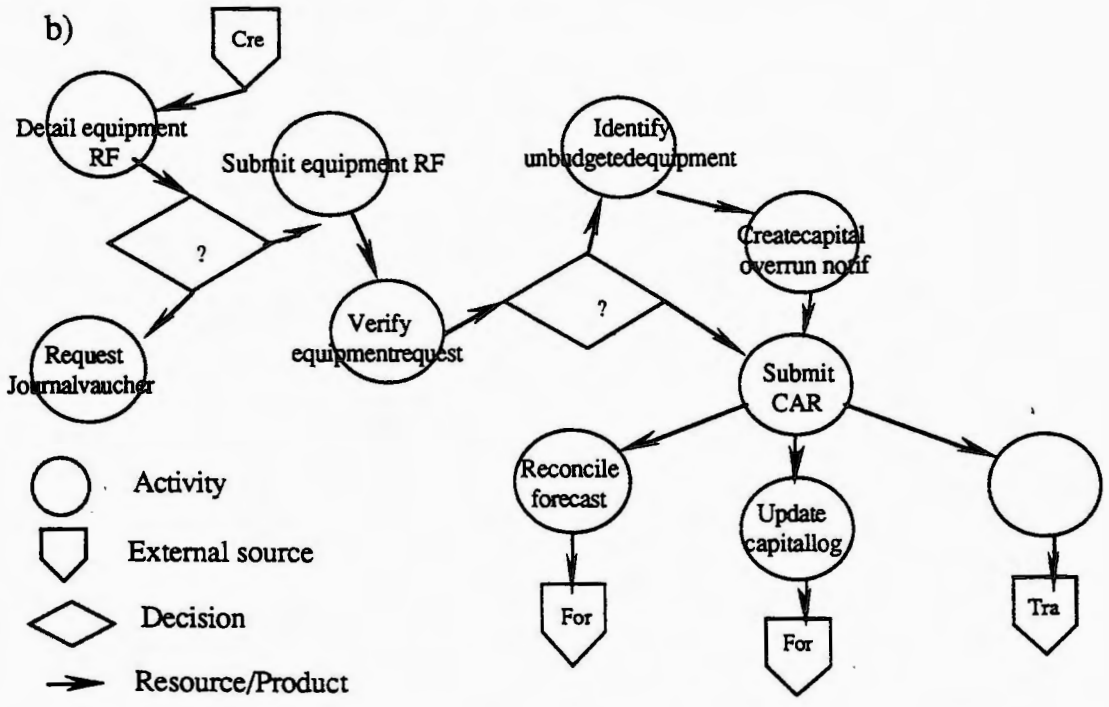


Figure 2. a) Tools currently under implementation for the Spark framework.
b) Part of a workplace description described with Flint.

3.4 The result of the study

We obtained a description of the task which is still being tested with tools under development by the DEC's group for effective application programming. Figure 2 shows what these tools are comprised of. This description has evolved into software artifacts useful to the Knowledge Consulting group resposible for the management of business information.

Overall we were able to generate a framework from our original questions, namely UnitedWorlds, to generate task descriptions that take into consideration both the task performer's and software developer's perspectives. In order to develop the framework we listened to taped interviews to identify which questions would have been helpful and what questions were not of any help. Some of the questions such as Q2.3 seemed interesting at first, but didn't provide any helpful insights for developing application programs. On the other hand, we looked at some of the tools presented in figure 2 and discovered that some information, required by Flint, was not being obtained and thus genereated other questions based on that lack of information. This framework is further explained in the following section.

4. UnitedWorlds

We have named our framework for task analysis UnitedWorlds because task performers are not treated as isolated problem solving machines, but rather are viewed as part of an open system from which they obtain resources needed during problem solving. Their relation to the inmediate environment is exploited to produce task descriptions that represent how they take part in their complex individual worlds. Those same relations are employed to connect those individual worlds with other individual worlds in a manner that checks for consistency and completness of the task description. This framework is best suited for cooperative tasks, but could also be applied to tasks thought to be performed by individual agents if it is recognized that those agents heavily depend on their interactions with the inmediate environment.

In this section we draw upon previous work done to classify manual interview questions (LaFrance, 1987) and introduce the idea that what is called in some cases "knowledge" should instead be named "answers" as explained in our next subsection.

4.1 Answer types

LaFrance makes a classification of knowledge in what she calls "Forms of Knowledge" (1987). We argue that what she classifies as knowledge type, as shown in the diagram of figure 3, is not knowledge, but the type of answers that knowledge engineers obtain from task practitioners. We don't really know how knowledge is represented in the human mind and can only guess it. Because of this it is more useful to attempt to classify how humans describe what they do than to classify knowledge we can't really see. Following this line of reasoning we have made a more extensive classification of possible types of answers one might obtain from task practitioners including some parts of LaFrance's original forms of knowledge classification. This classification is illustrated in figure 4.

4.1.1. Not-specific

We have chosen "not-specific" as opposed to "unspecific" as the the classification of this type of answers, because there are some useful specifics in the answer, but that are not focused to answering the question directly that sparked the answer.

4.1.1.1 Stories

LaFrance defined stories the following way: "Stories represent the classic cases and typical examples culled from the expert's long experience with the problem domain." We believe that stories are superficial representations that are indexed to knowledge by one or more factors in it. Task practitioners are usually good at telling stories of how they perform their task. It is sometimes particularly difficult to differentiate which stories represent the way human experts perform and which ones are fabrications of the way they think they ought to be performing. In any case, stories serve one very important purpose, that is, they are good to spark further questions aimed at discovering more specifics about how the task practitioner really performs the task.

QUESTION TYPES	FORMS OF KNOWLEDGE: LAYOUTS	STORIES	SCRIPTS	METHAFORS	RULES-OF-THUMB
GRAND TOUR					
CATALOGING CATEGORIES					
ASCERTAINING ATTRIBUTES					
DETERMINING INTER-CONNECTIONS					
SEEKING ADVICE					
CROSS-CHECKING					

Figure 3. LaFrance's Knowledge Acquisition grid. The horizontal axis represents "forms of knowledge" while the vertical axis represents "types of questions."

4.1.1.2 Metaphors

According to LaFrance "Metaphors encapsulate the expert's alternative images of the task, each of which includes unique features, constraints, and options. Metaphors describe one thing by reference to another apparently dissimilar thing so that the first is understood more completely than if the comparison had not been made." Metaphors are really a kind of story that refers to something else to make the point being made more clear. LaFrance also noticed that metaphors can be helpful to spark questions to obtain more specifics about the task. However, metaphors and stories have to be refined a lot before obtaining specific answers about how the task is performed. Metaphors as well as stories are in most cases volunteered answers from the task practitioners when they don't know how to answer in a more specific way.

4.1.2 Semi-specific, but needing validation

The following type of answers are more specific than the previous ones, but still need to be validated through other questions in order to check for consistency and completeness. For this reason, we classified this type of answers as "semi-specific, but needing validation."

4.1.2.1 Scripts

LaFrance writes: "Scripts give the expert's sequential and procedural knowledge of the domain... To know an expert's scripts is to have a temporal chart of critical actions, and to be able to understand each action terms of the prior knowledge required to perform it." This is an interesting description of scripts because it draws upon relation of knowledge and actions. However, it is not easy to obtain an script of how an expert performs a task or a part of it thereof. Therefore if a script is obtained from an expert on how he/she performs the task, it should be validated because it is prone to have many inconsistencies and not be very complete. Put in other terms, we suggest that if what seems to be an script obtained directly from a task practitioner, should be treated as a more specific story.

- TYPES OF ANSWERS: Answers provided by task performers during interview by task analysts.
  - NOT SPECIFIC: Answers with no particular focus given by task performers when they find it difficult to express directly what they do.
    - STORIES: Narative explanation given by interviewees of how they performed their task or at least thought they performed it.
    - METHAPHORS: Comparative narration making reference to another apparently dissimilar thing that helps understanding.
  - SEMI-SPECIFIC, needing validation: These are more specific but need to be refined and validated through other questions.
    - SCRIPTS: Sequential and procedural steps the interviewee takes in performing task.
    - RULES-OF-THUMB: Descriptive explanation of what situations or events cause some action.
    - DESCRIPTIONS: Qualitative explanation of some object or person.
  - SPECIFIC: Straight forward answers which are easily obtain becuase of the small size of their scope and because interviewees can easily relate to them.
    - ENTITY: Person, object or action.
    - ATTRIBUTE: Associated with entities.
    - VALUE: Of entities or attributes.
    - RELATION: Relations between entities.
    - LISTS: Of entities, attributes, values, actions, etc.
    - EXAMPLES: Specifc descriptions of instances.
  - TAXONOMICAL: Classificatory answers enumerating more specific or more general instances of some object, action, etc.
    - MORE GENERAL: More abstract generalization of something.
    - MORE SPECIFIC: More specific instances of something.

Figure 4. Classification of answers provided by task performers about their task.

4.1.2.2 Rules-of-thumb

LaFrance describes rules-of-thumb the following way: "Rules-of-thumb provide the myriad tactics and heuristics for interpreting and dealing with the array of circumstances encountered in carrying out the task." She further states: "Rules-of-thumb are concrete, implementable strategies of minor to moderate scope which can single out and define as issues those specific, limited conditions for which they serve as the most complete strategy." Described this way, rules-of-thumb look more like production rules used by a large number of expert systems to represent so called domain expertise than like something obtained as an answer to some question. according to our experience, it is very unlikely to come across answers from task

practitioners that look like rules-of-thumb. In the few cases we encountered such answers we found it necessary to refine them with other questions and found that some facts were missing and others were inconsistent.

4.1.2.3 Descriptions

Descriptions are very similar to stories, but are more specific because they aim at describing qualitative aspects of some object, person, relation or data item. Descriptions are different from scripts in that no procedural aspects are described.

4.1.3 Specific

Specific answers are straight forward answers that more easily obtained than previous answers. They comprise the following:

4.1.3.1 Entity

An entity could be a person, object or action, their more specific or their more general classification thereof.

4.1.3.2 Attribute

Attributes are usually associated with entities and can be obtained more easily when entities already have been identified. However, they are not as easily obtained. To ensure that enough attributes can be obtained directly from task practitioners, it is good practice to prepare a list of what such attributes can be.

4.1.3.3 Value

Values are probably the most easily obtained answers because they are related to something already defined in the task, such as an entity or attribute. Values can vary from string, Boolean or numeric to more specific ones such as real, integer and so on.

4.1.3.4 Relation

Relations are obtained when two different things of the same or different type are matched with each other. For instance, a person A can be related to action X by stating that A performs X.

4.1.3.5 List

Lists comprise sets or sequences of objects, actions, person, etc.. Lists are easily obtained from questions aimed at enumerating specific things that fall into some type already identified. But could as well be volunteered directly from the task practitioner.

4.1.3.6 Examples

Examples are different from metaphors in that they specifically describe an instance of a type already identified. In other words an example is like a member of a list that would be obtained if all examples were mentioned.

4.1.4 Taxonomical

Taxonomical or classificatory answers are a combination of list and relations. This type of answers enumerate more specific or more general instances of some object, agent, action, data item, Etc., into hierarchies.

4.1.4.1 More general

This type of answer is obtained when one or more things are abstracted into something more general.

4.1.4.2 More specific

This type of answer is obtained when something is more detailed into one more specific things.

4.2 Task layout operators

LaFrance lists task layouts in her "Forms of Knowledge" dimension and she describes them as follows: "Layouts incorporate the expert's 'map' of the task, including an understanding of its boundaries, organizations, and basic classification." We didn't incorporate layouts in our answer taxonomy because we considered that a task layout cannot be obtained directly from task performers and in the contrary is something that has to be constructed by the knowledge engineer exploring the task. We argue that "understanding" of task boundaries is pretty difficult to obtain,

but that instead layout operators can be used to "map" the task performer's understanding in an interactive way via interview questions proposed later.

The following set of operators is only intended to illustrate that such operators are useful to generate task layouts. In no way are they proposed as the unique set of operators and can be changed to Accommodate programming preferences. The operators we present here are the ones that the framework being used to test our questions uses, that is, the Spark, Burn and Firefighter framework. It is trough these operators that this framework makes a mapping to readily usable software artifacts.

4.2.1 Activities

Activities are verb/noun phrases that represent some action that a task practitioner performs. Activities are combined with other activities to produce activity networks that describe to great detail how some aspect of the task is performed. Activities are connected to other activities via data links that represent: the activity resource when it comes in as an input and the activity product when it goes out as an output. Activities can be performed by one or more agents and by what we call non-agents.

4.2.2 Agents

Agents are persons or group of persons members of the community of practice for a given task. Agents perform activities and interact with other agents in different ways, for instance, via electronic mail, shared programs, telephone, face-to-face, etc.. Agents can be abstracted into more general agents, such as design engineers, managers, and so on.

4.2.3 non-agents

non-agents are also tangible to some extent but opposite to human agents are considered to be inanimate things such as programs and databases or any other software artifacts that agents interact with and that behave as sole entities.

4.2.4 Data

Data is divided into resources and products for two reasons: 1) it is easier to relate the data directly to activities that use the data this way, and 2) it easier to check for inconsistency and incompleteness in the list of data items generated by task analysis. The reader will notice that almost all resources are considered as products by other activities an almost all products are treated as resources at some point. Only those resources that come from external sources from the community of practice are not products of other activities and only those products that go out of the community of practice to external destinations.

4.2.4.1 Resources

Resources are the data items consumed by activities.

4.2.4.2 Products

Products are data items produced by activities.

4.2.5 Vocabulary

All vocabulary used to represent activities, agents, non-agents and data is considered as vocabulary for the community of practice one or more its members. Vocabulary is used as index into software artifacts by a tool called Smoke in the Spark framework. Another tool called the Browser is used to describe all the vocabulary in a hierarchical manner.

4.3 Question types

LaFrance identifies a rich set of questions that are intended to help the knowledge engineer understand the task that an expert performs. However, she leaves out some important details, that is, what does the knowledge engineer's understanding of the task mean? Would two different knowledge engineers applying the same set of questions understand the task in the exact same way? Would the same question produce the same answer if asked in different occasions? How does one measure such understanding? We argue that it is necessary to introduce another dimension to LaFrance's grid, that is, the dimension of answer usefulness to overcome these issues. Figure 5 illustrates the classification we encountered while developing UnitedWorlds.

TYPES OF QUESTIONS: Extension of classification of questions asked by task analysts to task performers.	
TRIGGER QUERIES: Intended to probe the task being examined so as to spark answers that can be further examined later to obtain more specific ones.	
	GRAND TOUR: General questions aimed at obtaining no specific type of answer. E.g., "Describe what you do as an engineer."
	SEEKING ADVICE: Questions aimed at some particular part of the task. E.g., "What advice would you give me if I was to do market analysis?"
	NAIVE QUERIES: Questions aimed at some particular part of the task. E.g., "Why do you classify 'star' differently from 'cow' accounts?"
CATOLOGING CATEGORIES: The aim of this questions is to discover more general or more specific concepts in a taxonomy or classication.	
	QUERING FOR MORE GENERALITY: E.g., "Could you tell mewhat category does Expense Relief Voucher falls into?"
	QUERING FOR MORE DETAIL: E.g., "Are there different types of Telecommunications Customers?"
ASCERTAINING ATTRIBUTES: Questions aimed at discovering features of some object, person or action. E.g., "What is a Voucher composed of?"	
CROSS CHECKING: Questions aimed at evaluating how accurate or some answer might have been.	
	DETERMINING INTERCONNECTIONS: E.g., "You said only engineers filled Expense Relief Vouchers. Name all persons that fill vouchers.
	PLAYING DEVILS ADVOCATE: E.g., "You said that all proposals are double-checked by David. What if they weren't double-checked by him?"
	SEEKING THE EXCEPTION: E.g., "Can you think of some particular instance in which no Expense Relief Voucher was filled at all?"
	HOW SURE ARE YOU: E.g., "On the scale from 1 to 100, how sure are you that all customers must be billed by the end of the month?"
OBSERVATION: Observation, could be either passive or active. Passive observation can be complimented by requiring the interviewee to think aloud and later using information obtained during observations to ask more specific questions. While all question types, mentioned above could also be used during active interview.	

Figure 5. A classification of question types for workplace analysis.

4.3.1 Trigger questions

Trigger questions are those questions that produce answers rich with information useful to trigger other questions.

4.3.1.1 Grand Tour

Grand Tour questions as described by LaFrance are very helpful to get a sense of the overall task or tasks that the expert performs. However, they don't really help understand the boundaries of the task as the expert sees them, like LaFrance proposes. On the contrary, they just help as triggers for further questions that will then make task boundaries clearer. Grand Tour questions can be even more helpful if directed around task layout operators and if complimented with Mini-Tour queries for each operator kind thereof. We have experienced that some of the operator instances can more easily be extracted via Grand Tour questions than others. We believe that this is due to the abstract level which the task performer sees them. For instance, it is easier to answer questions about agents than it is to answer questions about activities. The former exist in space and time making it less abstract than the later which might only exist in time if it exists at all.

4.3.1.2 Seeking advice

Seeking Advice questions of the type proposed by LaFrance are good to explore still unknown areas of the task or domain but they are only useful in the sense that they spark other questions. Therefore, these type of questions can be considered as a useful subset of Grand Tour questions.

4.3.1.3 Naive questions

Naive questions, originally classified under cross-checking question by LaFrance, also produce answers that help trigger other questions. She proposed questions such as: "Bear with me while I ask what may appear to be a naive question. Could you tell me why orders need to be scheduled?"

4.3.1.4 Posing hypothetical situations

This type of questions also produces answers useful to trigger questions that seek more detail. Similar to naive questions posing hypothetical situations were originally classified as a cross-checking question.

4.3.2 Cataloging Categories

This is a very important kind of question because according to our experience it is very difficult to obtain instances for task operators which are at the same granularity level. So instead of deciding on the level, we have found that leaving the granularity open-ended is a more comfortable situation for the task performer and it is still useful to identify software artifacts that directly implement operator instances or group of them thereof.

4.3.2.1 Querying for more general things

When instances of task operators are extracted from task performers, it is useful to probe if there are natural generalizations of such operators that they feel comfortable with. It is sometimes more useful to find that the exact same output is produced for design engineers than to repeat the phrase: "The quarterly material budget is handed out to Bill." "The quarterly material budget is handed out to Paul." "The ..." Instead if provided that Bill, Paul ... are all design engineers, one could say: "The quarterly material budget is handed out to all design engineers in charge of the project." Where "all design engineers in charge of the project" represents a useful abstraction. Similar abstractions can be found for other types of task layout operators as well.

4.3.2.2 Querying for more specific things

Opposite to more general things, more specific things represent more detailed operators for a task description. Questions such as: "Could you tell me who are the persons that you classify under what you said were your managers?" Where "managers" is being explored for more detailed instances.

4.3.3 Ascertaining attributes

We found that it is helpful to partially identify the set of the attributes of layout operators and their possible values thereof, which we call the "form" and "content" respectively. The reason for this is that it is easier to predetermine them from pieces of already constructed software artifacts making it certain that they will be used for some purpose at all. However, we don't disqualify the use of the type of questions proposed by LaFrance when no such attributes exist or for the extraction of more specific domain knowledge.

4.3.4 Cross-checking

Cross-checking questions are very useful to validate answers obtained from other questions. We have reorganized LaFrance's questions a little bit and brought determining interconnection questions as a subtype of cross-checking questions and put naive questions and posing hypothetical situations as subtypes of trigger questions where they fit better.

4.3.4.1 Determining interconnections

We argue that determining the inter-connections should not just be directed at uncovering the relations among the concepts only in the domain, but also in the task. Everything should be related to something else. Nothing in a layout description should be completely independent of everything else because that means that it is not necessary at all.

4.3.4.2 Playing devil's advocate

Playing devil's advocate questions are very useful to identify exceptions and therefore cross-check answers provided by the task performers. One example that LaFrance gives is as follows: "Let me play devils's advocate in response to your story about the need to set priorities on scheduling multiple orders. What if you didn't set priorities?"

4.3.4.3 Seeking the exception

An example of seeking the exception question is: "You said that all proposals are checked by you and your boss John, in which instances a proposal is not checked by either one of you?" These are very useful to cross-check if given answers are complete enough.

4.3.4.4 How sure are you?

LaFrance proposes this type of question to cross-check given answers. However, the answer to this type of question can be very ambiguous unless some kind of measure or scale is presented along with the question. The scale at the same time must have some kind of use by software artifacts employed to implement the task or otherwise it would be meaningless to ask this type of question.

4.3.5 Observation

Observation also plays an important role, specially when it is possible to ask the task performer to think aloud when performing his/her task. When it is not possible for the task performer to think aloud, then observation can be used to spark other questions seeking information on why some actions observed was taken by the task performer.

4.4 UnitedWorlds's questions

Based on answer and question classification as well as task layout operators proposed, we were able to refine our original questions. The questions presented bellow are the result of this refinement. Each question type presents 1) a list of alternative questions, 2) answer type expected, 3) answer usefulness, and 4) useful spotlights questions, questions that inderectly aim at making clearer some concept via another related concept. Figure 6 cross matches these questions with the classifications presented above.

4.4.1 Questions generated from refining initial questions after listening to taped interviews

4.4.1.1 Functional description questions

1) Alternative questions

a) What are the functions your perform in your task? Enumerate as many as you can without leaving out things that might not be in your job responsibilities.

b) What contexts are there in your task? Enumerate as many as you can without leaving out things you think are not important.

c) Enumerate the topics of conversation you have with colleagues while performing your task?

2) Answer type

The goal is to obtain a list of functions the task practitioner performs. The functions in this list need not be at the same level of granularity. The attempt is not to obtain a complete list either, but to have a preliminary list that will later be pruned or completed. The answer from the task practitioner might be provided in a not-specific manner, that is they might be stories or metaphors. It is important that each of the functions be expressed in a verb/noun format.

3) Answer usefulness

The answer to this type of questions is very useful to break down the task being performed into smaller easy to handle chunks. The functions obtained serve as triggers to identify more specific actions that the task practitioner performs. At the same time software artifacts modularity might be designed around the function level with more modularity being applied to its more specific chunks.

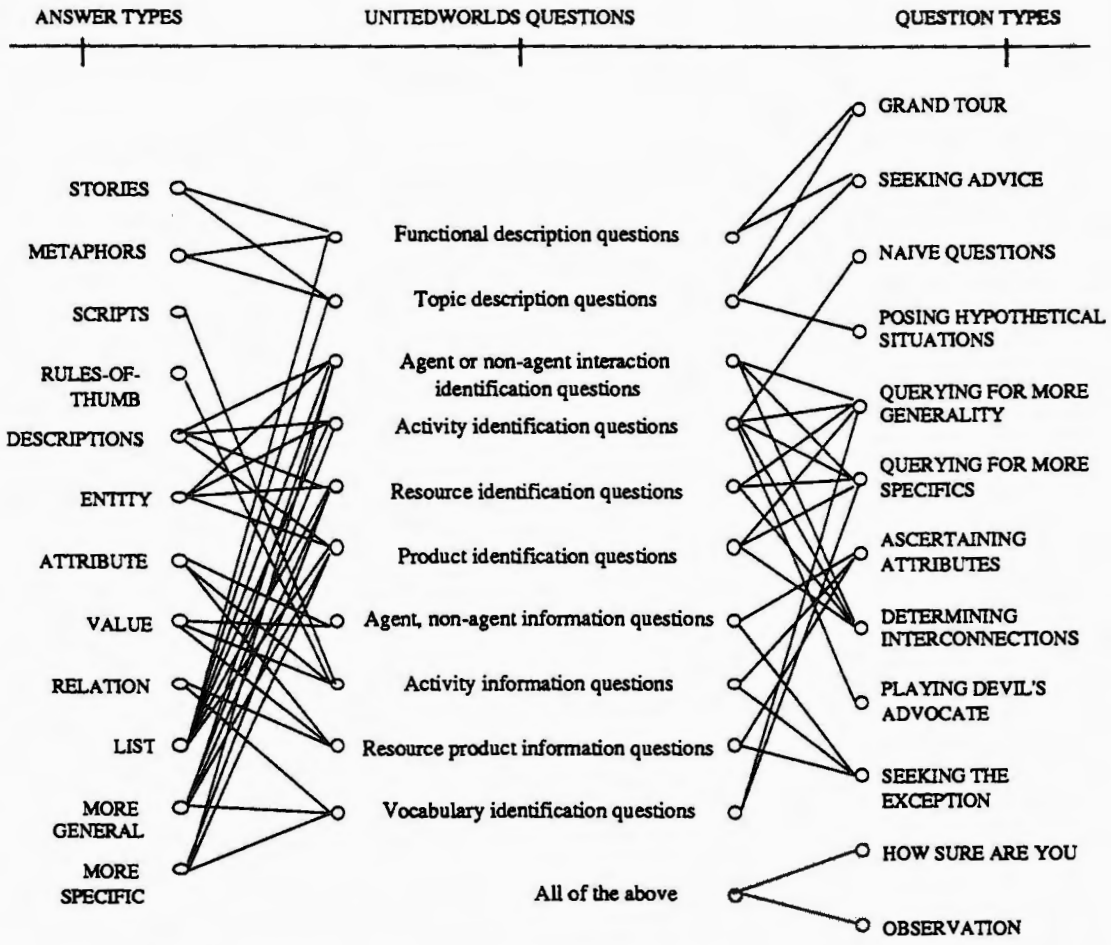


Figure 6. UnitedWorlds questions matched with answer and question types.

4) Spotlights

a) Activity grouping

This is a common brain storming technique to obtain function lists.

b) Purpose of interacting with agent or non-agent

It might be easier for some task practitioners to relate their functions to more tangible things such as other colleagues, programs they use to perform the task, data files, and so on. For this reason, one might want to identify these agents or things first and them group the type of interactions according to the agent's "feeling" into functions in much the same way it is done for activities.

c) Use topics to obtain more functions

The question might be: Is there anything else that you do in that topic?

4.4.1.2 Topic description questions

A topic is different from a function in that topics identify specific types of interactions inside the function. A topic can be contained in different functions and it is closer related to the domain type of knowledge. The agent might in some cases confuse topics with functions. The interviewer should be careful to make the distinction by himself/herself. For instance a function might be "proposal writing" while the topic might be "pharmaceutical industry," or "state and government." A topic describes more the domain part of the function and the function is more

like a subset of the task.

1) Alternative questions

a) For every function ask the following question: What are the different instances in which you perform that function?

b) For every function ask the following question: For what do you "function name?" Try to give as many as possible.

c) You say that you "function name" for "something." Do you do that for anything else?

d) You say that you "function name1" for "something." Do also "function name2" for "something"?

2) Answer type

The goal is again to obtain a list of topics for each function and then match it against all other functions to obtain a complete list of all topics the task practitioner performs in. The answer might need to be abstracted from not-specific answers or generated from function names given directly. For instance, the function "Roll up Expense Relief Voucher for industry A" could be used to ask: "Do you roll up Expense Relief Vouchers of other industries? The answer might come as not-specific, semi-specific or specific so the interviewer should be responsible to generate the tentative list for later validation.

3) Answer usefulness

The list of topics obtained is not only very helpful in identifying domain knowledge for the task, but also helps as a spotlight question to identify functions. Furthermore, it might be the case that topics determine exceptions for some functions.

4) Spotlights

Topic or subject of interaction with agents or non-agents.

4.4.1.3 Agent or non-agent interaction identification questions

Because agents and non-agents are tangible things, they are fairly easy to be identified directly from task practitioners. For this reason UnitedWorlds is based on identifying first all agents and non-agents with whom the interviewee interacts with. Then those interactions are exploited to identify other non-tangible parts of the task description.

1) Alternative questions

a) With whom do you interact with while performing your task? Enumerate as many persons as it comes to your mind that you think have been relevant to your task at least once.

b) With what do you interact with while performing your task? This might be programs, manuals, files, databases, etc.. Enumerate as many as come to your mind without leaving out anything you don't consider relevant.

2) Answer type

This questions provide a list of agents or non-agents. As explained previously, it should be fairly easy to obtain this list directly from the task practitioners because they are tangible and easy to be identified with. This list is what some social scientist call a "community of practice."

3) Answer usefulness

The list of agents obtained is useful to spark other spotlight questions to identify other layout operators in the task, such as to identify what is produced by and for each agent. Also if the purpose is to generate workflow managers for groupware, then this helps to identify who the agents should be connected through the workflow manager. Non-agents not only help in identifying other operators, but also help in making software implementation decisions because they implicitly indicate where data comes from or gets store into. In the ideal case this non-agents would be embedded in the resulting set of software artifacts.

4) Spotlights

a) If functions have been identified already, enumerate persons or non-agents you interact with in every function.

b) If topics have been identified already, enumerate persons or non-agents you interact with in every topic.

c) If activities have been identified already, identify who or what performs the activity.

4.4.1.4 Activity identification questions

Activities are non tangible and therefore are very difficult to be extracted directly from community of practice members. However, some questions may actually help get a tentative list of activities that must be pruned and refined to be of any help at all. 1) Alternative questions

a) Give verb/noun phrases of things you do while performing your task.

b) Ask Story generating questions and then extract all activities for the story by filtering verb/noun phrases. Then use them to extract other activities.

2) Answer type

The goal is to obtain a list of activities, but it will probably not be easy and not-specific answers might be obtained instead. The interviewer should then take those not-specific answers, stories, etc., and try to obtain as many verb/noun phrases from them.

3) Answer usefulness

Though difficult to obtain, these are probably the most useful operators when it comes to task automation because they seem to indicate the need for software artifacts. The mapping to software artifacts is made possible if all need attributes of the activity are instantiated. Spark uses activities for this purpose successfully and provides tools to instantiate their attributes in a flexible manner.

4) Spotlights

a) If resources or products have been identified, ask what activity(ies) consumes the resource and what activity(ies) produces the products querying for verb/noun phrases.

b) If Agents and/or non-agents have been identified ask what is done in the interactions with those agents querying for verb/noun phrases.

c) If functions have been identified, give verb/noun phrases of things you do to perform that function.

d) If topics have been identified, give verb/noun phrases of things you can do with those topics.

4.4.1.5 Resource identification questions

1) Alternative questions

a) What things do you need before performing your task in order to start performing it?

b) What things do you need while performing your task? Don't leave out anything that you may think is not important.

2) Answer type

Resources are usually tangible things, but opposite from agents and non-agents, may sometimes be intangible. Therefore, the interviewee may be able to identify some of the tangible resources easily, but it might be more difficult to provide the less tangible ones. Ideally the interviewee would be able to provide good-enough list of resources that can be matched later to products in order to refine them. The first of the questions may first produce a small list of resources, but if the second question is asked afterwards it might lead to larger list of resources that might be refined further.

3) Answer usefulness

The list of resources obtained serve many useful purposes among them: 1) they are useful to identify more activities, and validate identified ones, 2) They are useful to obtain domain knowledge, 3) they are useful to interconnect activities into networks that help define the layout for functions and tasks, and 4) they are useful to identify inputs and outputs for software artifacts.

4) Spotlights

a) If products have been identified already check if they are also used as resources.

b) If agents have been identified, ask what they provide to the interviewee as data that is relevant to perform the task.

c) If activities have been identified, ask what the activities consume as data or receive as input before being performed.

d) If Functions have been identified, ask what resources are needed to perform the function.

4.4.1.6 Product identification questions

1) Alternative questions

a) What things do you produce after performing your task?

b) What things do you produce while performing your task?

2) Answer type

The type of the answer is the same as that of resources.

3) Answer usefulness

Products are the same as resources but looked from the opposite perspective. So they are not only useful to validate resources, but also in the same way as resources are.

4) Spotlights

a) All spotlight questions used for resources may be used to help identify products, if the word 'product' is substituted for resource and vice versa.

4.4.2 Questions generated from the need that Spark has for information about task layout perators.

4.4.2.1 Agent non-agent information questions

1) Agent description questions

Ask questions such as "How would you describe that agent from your point of view?" This type of question generates a description or stories that can be translated into descriptions.

2) Interaction medium questions

For every agent or non-agent that the interviewee mentions ask what media of communication he/she employs to interact with him/her/it. Possible answers may be: 1) Face-to-face, 2) telephone, 3) fax, 4) e-mail, 5) voice-mail, 6) shared programs, etc.. This questions are particularly useful when software artifacts are being provided to the community of practice connected through workflow managers.

3) Interaction frequency questions

These questions deal with how often the interviewee interacts with other community of practice members. If the frequency of interaction is large compared to others, then it might mean that there are various activities that can be identified by exploiting the interactions with spotlights questions to identify activities, resources and so on.

4) Agent activities questions

If activities have already been identified then it is necessary to identify who or what perform each of the activities. This way software artifacts can be focused on the right agents or can be generated from already existing non-agents.

5) More general agents identification questions

In some cases agents might be generalized with terms such as "managers", "engineers" and so on. If that generalization exists it could be easier to refer to groups of agents. In order to find out such generalizations the interviewee must use terms that he/she is familiar with. These generalizations usually come natural and are very easily obtained as the interview goes on and the interviewee talks about agents. For instance, it might be easier for an interviewee to say that he does a part of his/her task by interacting with design engineers than by mentioning each of the names of the engineers. In any case, questions such as: "what is the position of agent1?" Followed with: "are there any more agents with that position?" can help identify such generalizations.

6) More specific agents identification questions

More specific agents are also easy to identify. When referring to agents the following question can be asked: "name all agents you interact with that hold that position of title." For our previous example one could ask: "you said that you interact with design engineers. Could you be more specific and name all design engineers that you interact with?"

4.4.2.2 Activity information questions

1) Activity description questions

Once an activity has been identified it is useful to obtain a description of such activity. Such a description will be helpful to document the activity and to trigger other questions that relate to the activity. By asking: "could you describe what you do when you perform that activity?" the user most likely will reply with stories, metaphors, descriptions or scripts. The last two are very desirable because scripts can help identify more specific activities or help design software artifacts and descriptions are useful for documentation.

2) Activity initiation questions

Ask questions such as: "what other activities determine when you should initiate this particular activity?" "When in relation to other activities do you start performing this activity?" Or: "What stimulus do you need to initiate this activity?" This is very helpful to position the activity relational to other activities, thus helping generate task layout descriptions in the form of activity networks.

3) Activity procrastination questions

Ask questions such as: "How long can this activity be postponed?" The aim is to determine an interval in which the activity could be started, if for some reason it couldn't be started right after the completion of other activities indicated that it should be started. This is helpful to give some procrastination allowance to the agents completing the activity.

4) Activity time-consumed questions

Ask how long it is expected to take to complete the activity once it has started. Alternative questions could be: "Estimate how long it usually takes to perform that activity once started," "How long do you think it would take you to finish that activity if you where to start now?" and so on.

5) Activity frequency questions

The answer to question is used to determine in what other circumstances the activity is performed. Possible questions are: "When do you repeat the activity?" "Do you repeat the activity in occasions other than the one you mentioned?" "What determines that you should repeat the activity?" and so on.

6) Activity agent identification questions

These questions include the spotlight questions mentioned above and are intended to determine who performs the activity. Other alternative questions include: "Who performs the activity?" and "Name all persons, group of persons or software artifacts that currently perform this activity?"

7) Activity resource/product identification questions

This questions also include spotlight questions mentioned above and are intended to determine what things are needed as input and what things are produced as output when that activity is to be performed.

8) More general activities identification questions

Activities are abstracts things and as such their granularity level varies from person to person. It is very difficult to find activities with the same granularity level. For this reason we propose that activities be allowed to have the granularity level adjusted according to the task from which they are abstracted. For this reason it is possible to navigate activities in two directions: more general activities and more detailed activities. More general activities are supersets of less general activities. To obtain more general activities from less general ones, one could ask such questions as: "Could you generalize this activity into some more general task?" or "In what context, task or function do you perform this activity?"

9) More specific activities identification questions

To obtain more specific activities it is simpler than to obtain more general. Questions such as "Could you give more detail of how you perform that activity?" or "What is it you do exactly when you perform that activity?" help identify stories, metaphors or scripts that can spark more questions to obtain more specific activities.

4.4.2.3 Resource/product information questions

These questions are important to interconnect activities based on input/output relations.

1) Resource-consuming activity questions

Resources and products are actually the same thing as you may have already guessed. We build upon this egalitarian relation not only to identify all data consumed or produced during problem solving, but also to validate such data. Questions needed to identify which activities consume a given resource (data item) could be phrased as follows: "What activities need this resource to effectively be performed?" or "Where is this resource used? Name all activities that come to your mind without leaving out anything you don't consider important." Spotlight questions such as: "Who or what uses this resource as input?" And then ask: "What activity does that person or software artifact performs with it?"

2) Product producing activity questions

Questions needed to identify which activities produce a given product (data item) could be phrased very similarly to those needed to identify resource consumers: "what activities need to be produced after it has has been effectively performed?" or "Where is this resource produced?" Name all activities that come to your mind without leaving out anything you don't consider important." Spotlight questions such as: "Who or what produces this resource as output" followed by: "What activity does that person or software artifact performs to produce it?"

3) More general resource/product questions

Resource or products can sometimes be generalized into more more abstract data items. This is helpful when more general activities than the ones currently consuming or producing the data items exist. Questions such as the following could help in finding such abstractions: "Can you group some of the resource/products that you have identified into more general categories?" "You said that the activity that produced or consumed this resource could be generalized into a more abstract activity. Could you also generalize the resources or products consumed so that the more general activity consumes it or produces it?"

4) More specific resource/product questions

Similarly to questions for producing more general resource/products, questions for producing more specific ones could read as follows: "Can you identify more specific resource/products for this data item?" "You said that the activity that produced or consumed this resource could be specified with this other more specific activities. Could you also specify the resources or products consumed or produced by those activities?"

4.4.2.4 Vocabulary identification questions

It is very difficult to ask questions about the vocabulary of the task directly to the task performers. Vocabulary represent abstract names we give to tangible and intangible things. Therefore to give a list of all vocabulary that pertains to some subject is to much too ask of task performers. Therefore all the instantiation of the task layout operators accomplished by all the questions described above result into vocabulary for the task and are treated as such when identifying software artifacts useful to automate the task or parts of it thereof.

4.5 UnitedWorlds Interview Strategy

We have found in our experience with interviewing that better results are achieved by arranging questions in a predetermined strategy that aims at fulfilling some goal. The goal might be to identify as many instances of one layout operator or at ordering them so as to discover problem solving behavior and so on. In order to generate an question strategy it is necessary to answer the following questions:

1) What is the goal of the strategy?

2) Which other operators best help to achieve the goal?

For UnitedWorlds, which exploits communication between task performers, we identified the following strategy.

Activities play an important role in describing how tasks are performed. However, it is very difficult for a task performer to describe directly how he/she performs the task. The reason, as we explained before, is that activities are not tangible things to the task performer. Therefore, we have designed an interview strategy that uses tangible things, namely agents, non-agents and data items, to indirectly identify what activities comprise the task.

1) Goal

The goal of the strategy is to identify activities performed by the task performer. Activities can be generalized into functions or into more detailed activities. Activities are connected through other activities via resources or products and form activity networks.

2) Operators that help achieve the goal

Agents, non-agents, resources and products are helpful. However, agents an non-agents are more helpful because task performers can more easily associate themselves with such operators.

3) Strategy

a) Identify as many agents as possible that the interviewee interacts with using agent or non-agent interaction identification questions.
b) Use functional description questions, based on agents and non-agents, to identify functions the interview performs in the task.
c) Use topic description questions and based on agents or non-agents try to identify as many topics as possible for the task.
d) Classify agents and non-agents into each of the functions identified.
e) Use agents and non-agents identified to help enumerate resources and products along with other questions independent on agents and non-agents.
f) Use agent and non agent information questions to find out more about the agents non-agents and to cross-check activities they perform.
g) Use resource/product information questions to find out more about and cross-check resources and products and the activities they are consumed by and produced from.
h) Use activity identification questions based on all previously obtained answers to identify as many activities as possible.
i) Use activity information questions to find out more about the identified activities and to cross-check them for consistency and completeness.
j) Try to interconnect the activities obtained via their resource and product relations.
k) Refine the activity networks obtained based on functions, topics, agents or non-agents, resources or product related questions.
l) Repeat any of the above as necessary.

5 Conclusions

One of the main motivations of this study was the belief that active involvement of task performers would increase the success of applications that support their task which is a widely shared view with social scientists and some computer scientists as explained in the book Design at Work edited by Joan Greenbaum and Morten Kyng (1990). This is a human factor that has been greatly ignored in the knowledge-based system development community and has caused failure of a large number of systems because its users feared to use them due to lack of understanding. With the UnitedWorlds approach we hope that the gap will be filled more between experts and knowledge-based systems because the experts will be able to not only describe their task but also handle the resulting system from the description.

Overall we were able to generate questions and questions strategies that are helpful in describing cooperative task. The answers obtained from task practicioners can be used to generate software artifacts that fit rather than fit on the workplace. This questions are still under refinement

and are being tested by DEC to help generate task description for use by the Spark, Burn and FireFighter framework.

References

Dallemagne et al. (1992) Making application programming more worthwhile, *In Knowledge Engineering and Cognition*, F. Schmalhofer and G. Strube, eds., Springer-Verlag, Berlin.

Greenbaum, J. and Kyng, M. (1991) *Design at work.* Lawrence Erlbaum Associates, Publishers. Hillsdale, N.J.

Klinker, G et al. (1991) Usable and reusable programming constructs, *Knowledge Acquisition*, Vol. 3, 117-135

LaFrance, M. (1987) The knowledge acquisition grid: a method for training knowledge engineers. *International Journal of Man-Machine Studies*, **26**, 245-255

Marques et al. (1991) More data on usable and reusable programming constructs, *Proc. Sixth Knowledge-Acquisition for Knowledge-Based Systems Workshop*, SRDG Puplications, Dept. of Computer Sci., Univ. of Calgary, Alberta, Canda, T2N 1N4, 14-1 to 14-19

Marques et al. (1992) Easy programming - empowering people to build their own applications, *IEEE Expert*, Vol. 7, No. 3, 16-29

Mizoguchi, R., Tijerino, Y., and Ikeda, M. (1992) Two-level mediating representation for a task analysis interview system. *In Proc. of AAAI-92 Worshop for Knowledge Representation Aspects of Knowledge Acqusition.* San Jose, Ca., 107-114

McDermott J., Dallemagne G., Klinker G., Marques D. and Tung D. (1990) Explorations in how to make application programming easier. *In Proceedings of 1st Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop.* 134-147

Tijerino, Y. A., Kitahashi, T. & Mizoguchi, R. (1990) A task analysis interview system that uses a problem-solving model. *In Proceedings of The 1st Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop*, 331-344

Tijerino, Y. A., Kitahashi, T. & Mizoguchi, R. (1992) MULTIS: A knowledge acquisition system based on problem solving primitives. To appear on *International Journal of Expert Systems*. Also in *Proceedings of 6th Knowledge Acquisition for Knowledge-Based Systems Workshop* 32-1 to 32-20

DESIGN ISSUES FOR DESIGN KNOWLEDGE ELICITATION

Andrew Tunnicliffe
Department of Computing,
De Montfort University,
LEICESTER, LE1 9BH.
UK.
at@leicp.ac.uk

Stephen Scrivener
Department of Computer Studies,
Loughborough University of Technology,
LEICESTERSHIRE, LE11 3TU.
UK.
s.a.scrivener@lut.ac.uk

Abstract : Insufficient effort has been applied to research into techniques for design knowledge elicitation. We maintain that it is necessary to consider the nature of the design process more carefully, and take account of important design issues in order to develop credible methods for design knowledge elicitation. Reductionist views of scientific problem-solving are currently being applied in the attempt to develop design knowledge based systems, with scant regard to the broader issues of design, failing to observe that design theoreticians and practitioners offer alternative accounts of design activity and that design must be considered dissimilar to scientific problem solving. In particular hierarchical decomposition and bottom-up construction are not generally accepted views of design activity. Spatial, diagrammatical and drawing forms of communication, that are manifest in design, must be tackled. Design tasks that involve visual design skills appear especially neglected, and we have particularly targeted these domains for study. We propose a holistic approach to design knowledge elicitation that is both more relevant and pragmatic.

Introduction

This paper is one of a series discussing design knowledge elicitation and reporting our recent studies (Tunnicliffe 1990). Our approach, in this paper, is to review the nature of the design process in order to promote better methods of design knowledge elicitation. The details of the study, the methods used, and verification procedures are subjects to be discussed elsewhere. In this discussion we put forward characteristics of design, highlighted by design researchers, that we believe need to be addressed by researchers of design knowledge elicitation and argue that an approach that is altogether more global, and holistic is appropriate.

It is widely accepted that design is difficult to study (Lawson, 1980). Cross et al (1981) argue that "commonly-adopted 'scientific' models of design are not really tenable in view of the current epistemological chaos surrounding the concept of 'scientific method' ... design simply is not like science." In a review of design methodology literature, Cross (1984) concludes that it is unlikely that design methods can be constructed as comparable counterparts to scientific methods.

Landsdown (1988) reviews evidence that suggests designers consistently ignore, or fail to apply, established design techniques, and argues that "There is considerable evidence to support the observation that many designs are executed without necessary and relevant information being taken into account (See for instance Whyte, 1975; Scott, 1976; Bignell, Peters and Pym, 1977; Allen 1984; Bignell and Fortune, 1984). Indeed, it is clear that most design failures arise not because designers are working to the boundaries of current knowledge

in their particular disciplines. They generally arise because designers have not employed well-understood and often well-documented principles, procedures and practices. They have simply not made use of existing results (BRS, 1978)." Of particular concern are the areas where safety is paramount (Nader, 1965; McConnell, 1987; Stewart, 1988) and where the financial impact of failure in high-cost applications is substantial. The domain of Aerospace (eg McConnell, 1987) being one in which knowledge elicitation has been extensively applied.

Smithers et al (1989) argue that, "Design as an intelligent behaviour is not well understood, and in some respects it is hardly understood at all. This lack of understanding presents severe difficulties when trying to build artificial systems for anything but relatively small clearly understood design tasks. Also, the techniques we have available for building such systems makes it very hard for us to build anything but limited systems."

Thus, it is clear that considerable effort is still required in order to develop credible Design Knowledge-Based Systems, and a deficiency in performance is observed (eg Landsdown and Roast, 1987; Akman et al 1988). The scarcity of proven design knowledge elicitation procedure has not dampened the enthusiasm for "Knowledge-Based" Design Systems (eg Gero, 1989). The lack of design knowledge elicitation procedure is noted by Magee (1987), and Landsdown (1988) argues that deficiency of knowledge based design systems "... arises from the paucity of good tools for knowledge elicitation, acquisition and evaluation." We maintain that design knowledge acquired from ad hoc, unsubstantiated and untested procedures, and knowledge that is undocumented and untested cannot be considered reliable. We further argue that design knowledge elicitation has not been adequately addressed.

CHARACTERISTICS OF DESIGN

Features of design

Archer (1991) identifies that design is a process directed at meeting a particular need, to invent, construct or arrange something that had previously not existed. Archer (1991) argues that, in contrast, the process of science "seeks to isolate a phenomenon from the complexities of the situation in which it is embedded, and to abstract generalisable principles from observation and experiment." Archer (1991) recognises a number of features of design.

Useful : The products of design have purpose.

Productive : Design is active (science is explanatory).

Intentional : Design is not the discovery by chance, and designing requires many judgments to be made on complex issues, often utilising imperfect information.

Integrative : Design knowledge can be drawn from any and every field of endeavour, wherever appropriate to the task at hand, and as such is unbounded.

Inventive : Design involves the discovery of something new.

Expedient : Design products justify design activity. In contrast to science, the rigour of the application of methods or procedure is not fundamental to the design process, but validation of a design rests firmly on the results meeting requirements.

Modes of thought in design

The process of design may be considered less constrained than scientific problem solving as it involves a greater variety of information processing (eg textual, spatial/diagrammatic), a broadness of processing paradigms (eg diversification, focus) and encourages a full contribution of thinking style (eg impulsivity, reflectivity). For example, Cross et al (1981) argue that "designing relies heavily on modes of thought and ways of knowing which (so far) are incompletely defined and poorly understood, but which are characterised by being neither 'scientific' nor 'literary'."

The design methodologists (eg. Chrisopher-Jones, 1963; Archer, 1963) attempt to teach designers to broaden their preferences and make use of the spectrum of thought processes needed whilst designing, or encourage them, practically, through the provision of design methods.

Citing evidence from Education, Learning, and other research literature, Cross and Nathenson (1980) argue that people show preferences towards identified styles of thought, that these distinctions can form valuable contributions to cognitive-based design theories, and practical implications for design education are raised. The parallel between design education and design knowledge elicitation is clear, as knowledge elicitation can be viewed as a learning exercise (eg Johnson and Johnson, 1987). The impact of modes (and styles) of thought with reference to design knowledge elicitation will be discussed further in the section Design Issues for Design Knowledge Elicitation.

Cross and Nathenson (1980) discuss four dichotomies of cognitive style emerging from educational psychology that are of special interest to design. It is important to note that people tend towards one pole of a given dichotomy, as opposed to being either one or the other. The four dichotomies discussed by Cross and Nathenson are :

Divergent / Convergent : Divergent thinking involves the postulation of many alternatives, and convergent thinking tends towards a focus, for example, on a single 'correct' solution.

Impulsive / Reflective : An Impulsive style of cognition is where an immediate response is of highest priority - the first answer that comes to mind - whereas a Reflective style involves a degree of deliberation before an answer is ventured, the correctness of response being of highest importance.

Field Independent / Field Dependent : Field-dependent perception is significantly influenced by the surrounding context of objects, Field-independent perception allows the consideration of objects in a way that is less constrained by particular context.

Holistic / Serialistic[1] : Serialists proceed by small steps and attempt to establish each point

[1] The Serialistic and Holistic Dichotomies are experimentally observed by Pask and Scott (1971). Pask and Scot (1972) conduct a series of experiments to identify the educational impact of learning and teaching preference. A number of students are matched and mismatched against holistic and serialistic learning styles. In evaluation, it is shown that effective learning results when the students are matched in terms of learning style (ie holist/holist and serialist/serialist), but there is a failure when mismatched (ie holist/serialist and serialist/holist).

before moving to the next, without digression. They learn and remember string-like structures that are initially linked by low-level relations. The recall and teaching abilities of serialists directly reflects the particular mechanisms and order in which information is originally acquired. Holists approach tasks (and learning) in a less logical fashion, grasping and remembering information from various perspectives. High order relations are (initially) used by holists to remember whole structures. Explanations may take many forms, reflecting the multiplicity of the holist learning style, and concepts may be taught using a different method and in a different order to that when originally learn.

It is interesting to note that designing benefits from a wide range of these styles, but at appropriate stages of the design process (Cross and Nathenson, 1980; Cross, 1982; Tovey, 1984). In contrast, scientific dogma tends to undervalue (but not rule out) the Divergent, Impulsive, Field-Independent and Holistic forms of cognition, that are recognised to be of high importance to design.

Recently, design researchers (Edwards, 1979; Cross, 1984; Ward, 1984; Tovey 1984) have argued that results emerging from the field of neurology could offer pertinent explanations of observations emerging from the field of design. These researchers suggest that spatial design skills and 'creative' and holistic styles of thought may be centred in the right hemisphere whilst the linguistic, sequential and analytic (organisational) processes appear to be dominant in the left hemisphere. The issue of brain laterality is not discussed further in this paper.

Preferences in design

Darke (1979) argues that design is a process of reducing a large number of possible solutions by external constraints imposed by the particular design task, and internal constraints imposed by the designer's own thought processes. Termed the *Primary Generator* - an adaptation of the *Generator-Conjecture-Analysis* model of Hillier et al (1972) - an initial design solution is rapidly developed through a tight, nearly inseparable analysis and synthesis process recursively applied to the problem. Thus, a large number of possible design solutions can be extensively pruned by a small piece of information, a personal preference, or an arbitrary decision, in early stages of design.

Thomas and Carroll (1979) argue that design is a class of problem that is somehow ill-defined in the goal-specifications, the initial conditions, or the allowable transformations. Therefore, Thomas and Carroll (1979) argue that although there are common features of the design process, irrespective of domain, there is an inherent need to specify goals whilst designing, and the activity greatly depends on individual interpretation of those goals. Further, Thomas and Carroll argue that although design problems appear to be structured into sub-problems, the sub-problems themselves cannot be specified initially, and are dynamically produced through the process of design.

Spatial information and diagrammatical information in design

Ferguson (1977) highlights the value of images, visualisations and pictorial modes of thought in design. The value of drawing in design is well known, for example, Landsdown (1987) argues that "Drawing in all its various forms, from freehand sketching to detailed technical layout, is a type of modelling that designers find indispensable. In many cases, indeed, drawing is the only form of external modelling a designer uses. It has two basic functions: to assist in the *externalisation* and development of mental concepts and to help in the

presentation of those concepts to others." The impact of the spatial contribution in designing is also highlighted by Ward (1984) and Anita Cross (1984).

To conclude this section, Carroll and Rosson (1985) identify four fundamental characteristics of design :-

Design is a process : It is not a state and cannot be adequately represented statically.

The Design process is non-hierarchical : It is neither strictly bottom-up nor strictly top-down.

The process is radically transformational : It involves the development of partial and interim solutions that may ultimately play no role in the final design.

Design intrinsically involves the discovery of new goals.

WHAT DESIGN IS NOT

Design is not problem solving

The first major point of this section is that design researchers argue design is different to scientific exploration. For example, in reviewing the process of design, Cross (1982) concludes that there are 'designerly' ways of knowing, and "There is a distinct 'designerly' form of activity that separates it from typical scientific and scholarly activities." Although firmly a reductionist, Gero et al (1985) admits that "Architecture lacks the scientific tradition of disciplines such as Chemistry, Medicine or Geology."

Lawson (1972, 1979) establishes experimentally that science and design students apply different approaches to solving a problem. Final year science students use a problem-focused approach. The scientists define and understand the problem first, and then propose solutions. However, (Architectural) Design students use a solution-focused approach. They generate a solution to the hidden problem first, and then attempt to define the problem. Lawson, in a further set of experiments using first year students, demonstrates that the strategies are learned. The first year science and design students fail to adopt any specific strategy, and the performance of both the science and design groups are far less successful than their final year counterparts.

Support for this solution-focus in design comes from a recent study of group design (Scrivener et. al., 1992). In this study, which concentrated on communication patterns and media usage, pairs of student designers were asked to engage in design tasks. Their activity was recorded using audio-visual methods and subsequently coded and analysed. One of the methods of coding verbalisations, progression analysis, involved categorising utterances as being about a problem, a solution, or the process of designing. Analysis of this data showed that solution-focused utterances represented 60% of the total, with only 29% of utterances being problem focused.

It could be argued that the difference between scientific and design problem solving is simply the result of a perpetuated training method. However, design teachers are usually intelligent practitioners whom, we can assume, have applied their skills reflectively and critically, thus suggesting that design problems are fundamentally different to scientific problems, requiring

different solution or resolution strategies[2] .

Bijl (1985) argues that "design is not problem solving ... our thinking is too much conditioned by established concepts associated with successful problem oriented systems." Further, in support of Holt (1985), Landsdown and Roast (1987) argue that reducing all designing to problem-solving is too narrow a view, and conclude that "the whole of designing can not be equated with problem solving."

Design is not hierarchical Top-down decomposition or Bottom-up-construction
Luckman (1967) argues that the combination of interdependent sub-solutions cannot take into account the complex interrelationships of design. Luckman proposes that a more holistic view incorporating a domain of compatible sub-solutions, termed a 'decision area' is more appropriate. A decision area is an elaboration of all possible (design) options, to a given degree (the degree is the parameter of the decision area), so that all compatible options can be explored.

Bijl (1985) compares and contrasts the differences between design and problem solving activity and observes that a design cannot be readily decomposed into parts, and that there is a 'wholeness' in design, admitting "if we recognise the existence of parts, we do not know how to define them as discrete parts. It is not practical to work with discrete parts where changes to one part are likely to propagate unforeseen changes to many other parts." Bijl (1985) further argues that the process of analysis "is to decompose a problem into parts until individual parts are recognised as being amenable to known operations, and results are reassembled into a solution. This process has a peripheral role in design."

Some empirical evidence against the notion of design as decomposition comes from Scrivener and Palmen (1991). From an analysis of the drawing behaviour of designers working in pairs on the conceptual phase of the design, they argue that the designers do not decompose the problem (eg. one design the legs of a portable barbecue, the other the tray), instead they seem to divide the process (ie. one appears to be predominately a synthesiser, the other an appraiser).

Therefore, design cannot be generally described in terms of the function of ever smaller sub-components. The reductionist theory of scientific analysis, diagnosis, classification, hierarchical decomposition or bottom-up construction appears less appropriate when applied to design problems, in contrast to gestalt theories which appear to offer more promise.

Design is not Analysis
Design is not a process of analysis, explanation, or classification. Cross et al (1981) identify

[2] There is also anecdotal evidence (Scrivener et. al. 1992) that the solution-focused nature of designing is not simply the result of training in design methodology. In studying designers working in contexts of impoverished communication (ie no visual or verbal contact, other than writing) some pairs Scrivener has observed to quickly develop protocols which preserve much of the interchange occurring in communication rich situations, whereas others seem to exhibit a pattern of working which can be understood as an analysis, synthesis, appraisal sequence. In this latter case, where the strategy seems to be one in which the design pairs apply a theoretically correct method of designing, the solution appears less well developed than the former, thus suggesting that the solution-focused method of designing is discovered through practice of designing.

the predominance of 'tacit' design knowledge, and argue "...*knowing that*, ie. knowledge of the explicit 'rules' of design can actually inhibit practise." In design, the designer contributes additional information not explicit in the design objectives (Archer, 1979) and such information may have dramatic effects on the eventual design solution. The additional information may take the form of, but is not limited to, personal preferences, style and knowledge of the real World. Thus, Bijl (1985) argues that "It is characteristic of design that both the process and the product are not subject to explicit and complete criteria. This view of design differs sharply from the more orthodox understanding of scientific and technological endeavours which rely predominantly on a process of analysis." Bijl (1985) further argues that "the absence of well-defined and widely recognised criteria for design excludes it from the main stream of analytical developments."

Archer (1979) concludes that "It is widely accepted, I think, that design is ill-defined. An ill-defined problem is one in which the requirements, as given, do not contain sufficient information to enable the designer to arrive at a means of meeting those requirements simply by transforming, optimising, or superimposing the given information alone ... 'the problem' in a design problem is not the statement of requirements. Nor is 'the solution' the means ultimately arrived at to meet those requirements. 'The problem' is the obscurity about the requirements, the practicability of envisageable provisions and/or misfit between the requirements of provisions."

DESIGN KNOWLEDGE ELICITATION

So far we have discussed pertinent characteristics of design (and what design is not). In this section we examine the current approaches to design knowledge elicitation, outline important considerations to address the implications of design issues, and propose an appropriate 'holistic' approach to design knowledge elicitation.

Approaches to design knowledge elicitation

Characteristics of the design activity have been identified by design researchers and others that aptly distinguish design from planning, diagnosis and problem-solving. Current views of design knowledge acquisition view design, hence design elicitation, as hierarchical top-down decomposition (Chandrasekaran, 1988; Buck, Clarke, and Lloyd, 1991) and bottom-up construction (Marcus, 1988) and apply the reductionist approach of science. It is argued here that design research offers alternative views of design strategy, most notably the holistic nature of design (Cross and Nathenson, 1980; Bijl, 1985). Kornell (1987) identifies two kinds of thought (narrative and formal) and argues that knowledge elicitation tools fail to take account of differences between them. It is observed here that narrative thought appears more closely associated with design thinking (ie, via metaphors, analogies and gestalts), and formal with that of science and problem solving.

The domain of design, 'Routine design', as addressed by Chandrasekaran (1988), does not conform with the intuitive and reported characteristics of design discussed previously, and in our view, does not provide an acceptable conception of the broad process which is understood as design by designers (eg Darke, 1979; Thomas and Carroll, 1979; Carroll and Rosson, 1985). 'Routine design', where constraints are known and well defined, appears to be closer to a problem-solving planning task (Tate, 1985) than a design task. Further, Dreyfus and Dreyfus (1986) post a telling argument against addressing severely limited problems, where real difficulties have been excluded in the belief that they can easily be overcome with

developments that progressively ease the restrictions. It is unlikely that the exploration of 'Routine design' will offer valuable cornerstones for the building of an understanding of creative design, as is implied by the classification of the four types of design ranging from routine design through to creative design[3]. It is argued here that a broad (holistic) approach is required, which does not impose restrictions in order to transform the underlying problem being addressed (ie design) into another apparently more tractable one (ie planning).

Approach / Method	Authors
Automated Repertory Grid	Boose and Bradshaw (1987)
Propose-and-revise	Marcus and McDermot (1987)
Top-Down Decomposition	Chandrasekaran (1988), Buck, Clarke and Lloyd (1991)
Bottom-up component level (piecemeal) Construction	Marcus (1988)
Protocol Analysis	Johnson, et al. (1988)
Induction	(From Quinlan, 1987) eg. Kamal et al (1989), Guena and Zreik (1989)
Diagrammatic	Johnson and Johnson (1987) Li and Bourne (1989) (also Johnson et al, 1988)
Corporate Memory	Boose, Shema and Bradshaw (1990)
Preference *	Bradshaw and Boose (1990)
Prototypes	Alem and Maher (1991)

Table 1 : Recent approaches to design knowledge elicitation.
(* not specifically targeted at design)

The prototype approach to design solutions (Alem and Maher, 1991) can be positively viewed as solution-focused, since a solution of some description is proposed, and tested. This general approach is also central to design problems that utilise propose-and-revise strategies (Marcus and McDermott, 1987). However, the discovery of constraints is believed to be integral to the ill-defined nature of the design process (Simon, 1973) and as previously discussed, Archer (1979) argues that superimposing, optimising or transforming requirements do not sufficiently account for the process of design. Therefore, the value of prototyping from a given set of

[3] "The framework suggests that design by decomposition (ie breaking problems into subproblems), by plan selection, and by plan synthesis (as a last resort) are the core processes in Knowledge-Based design. This suggests an informal classification of design problems based on the difficulty of these subtasks or processes." (Chandrasekaran, 1988).

requirements and known operations also appears of limited use, and restricted to a set of design tasks that do not themselves include the discovery of new design goals, requirements or constraints during the process of design.

Marcus (1988) argues that a computer based framework can enable designers to describe higher level structures in a bottom-up fashion from components. The approach is based on the argument that basic components can be most easily identified first, and then the task of progressively organising and structuring the information can follow.

We suggest that the bottom level of decomposition may not be immediately obvious, and identifying components may not be simple or meaningful to the designer. We further argue that whilst designers may be able to identify components, they may not be able to provide valuable information of how such components interact at that (component) level. The interaction may be too complex or involve perceptual processing, or the designers are just not conceptualising the problem at that level whilst designing. The value of recognising component parts whilst not actually designing may help to identify the structure of the domain, but is unlikely to provide valuable information on the interactions, in design. Indeed, Marcus (1988), acknowledges that "[Designers]...are less clear how the individual considerations fit together in a step-by-step procedure for producing a design. The main difficulty is that decisions affecting one part of the design often impact other parts of the design in complex ways". For example, in the domain of page layout design (Tunnicliffe, 1990), the components *Headline text, Body text,* and *Pictures,* were easily identified, as were terms *weight, balance, colour, impact* and *emphasis,* that apply to the components. However, it was not possible to elicit reliable information about detailed interactions of components, with respect to the identified terms. In contrast, broad strategic knowledge was more readily acquired. We agree with Johnson et al (1988) that to acquire accurate design knowledge, designers must be engaged in design tasks.

Automated methods of design knowledge elicitation are gaining popularity for knowledge based design systems developers (eg Kamal et al, 1989; Guena and Zreik, 1989) seemingly promoted by the abundance of automatic elicitation methods appearing in knowledge elicitation research literature (eg Boose and Bradshaw, 1987; Quinlan, 1987). However, automatic elicitation methods seem to have been embraced from the domains of scientific problem solving, analysis, or classification, without adaptation, and failing to observe the special problems and requirements posed by design. For example, design cannot be equated to classification and analysis, the domains most suited to the above methods.

Boose, Shema and Bradshaw (1990) define design knowledge "... as all physical descriptions of a system and its components; rationale for design decisions; functional flow diagrams; documentation of design objects or processes they perform along with the results; and interrelationships among design knowledge elements (such as part numbers and descriptions)." Boose et al (1990) discuss an approach by which considerable (design) information is stored during the lifetime of a project (corporate memory), in order to facilitate later redesign and evaluation of alternatives in the domain of engineering trade studies. Although this research team identifies the need to address alternatives in design (Shema, Bradshaw, Covington and Boose, 1989), our understanding of the term design, and hence design knowledge elicitation differs.

It is also worth noting that the areas of design that we are investigating, where spatial and

visual types of design knowledge are utilised directly in the design task, are subtly different to the domains of design previously reported (Table 1).

Design Issues for Design knowledge elicitation

It is postulated that considerable design information is available to methods of knowledge elicitation. The identification, structuring and modelling of determinants of design may yield useful design information, and provide theories for focused empirical study. Evidence that designers can communicate expertise is demonstrated by the 'apprentice' approach to design education and practice. Gero et al (1985), for example, term this form of expertise transfer 'Design Lore'.

The need to elicit, and explicitly model preferences has been recognised (elsewhere) to be an important consideration for knowledge engineering (Bradshaw and Boose, 1990; McSherry, 1991). Magee (1987) argues that it is important to acquire "attitudinal" design knowledge. As discussed previously, design is often greatly affected by personal preferences (Darke, 1979), and methods of design elicitation must address the acquisition of preference-based knowledge. It becomes an important aspect of design knowledge elicitation to attempt to rationalise aspects of designing that are subject to preferences, and aspects that are not. This task is not simple, and we have no quick-fix solutions. However, our studies (Tunnicliffe, 1990) have indicated that using many experts in both elicitation and evaluation, the development of a deep understanding of the domain will enable preference-based knowledge to become distinguishable from domain knowledge that holds a consensus of generality amongst the domain experts[4].

Cross and Nathenson (1980), argue that design requires a subtle combination of a number of 'modes of thought', important at different stages of the design activity. An expert will adopt personal preferences (of cognitive style) for both problem solving and for explanation (ie teaching). The two may be quite distinct and the information obtained when in problem solving may be completely different to instructional explanations. Also, the knowledge engineer will behave with a cognitive style (preference) that may not match the expert's. Thus, two important issues for design knowledge elicitation are raised.

[4] In the above study four design experts participated in the first phase of the knowledge elicitation study, and one of these (E0) was extensively utilised in a second stage. Through the participation of these four experts, preferences of E0 were identified as a contrast against the other three. The accuracy and validity of the study as a whole was evaluated through a separate evaluation phase involving ten design experts (judges) having no previous involvement with the study, plus E0, the main expert of the knowledge elicitation study. Prior to the evaluation, predictions over E0's preferences (and other judgments primarily the focus of the evaluation) were made. The acquired knowledge, that predominantly represented the preferences of E0, was tested against the ten independent judges, and also against E0. The judges were shown to be in general agreement, although some stark differences were present. Therefore, in analysis, where E0 and the predictions relating to the acquired knowledge of E0 corresponded, but where the independent judges universally disagreed, then such knowledge can be clearly viewed as a preference of E0. Similarly, preferences of E0 can be recognised as such when the predictions attain contrasting agreement and disagreement from the judges. Additional identification of preferences of E0 and also of the judges can be identified where wide disagreement between the judges results, but where a failure to predict a general consensus was present (ie where no forecast of a preference was made, but shown to be so in the evaluation).

Firstly, in knowledge elicitation, expert designers must not be forced into a cognitive style (or mode of thought) that is intrinsically and diametrically opposed to their preference[5], is unnatural to their usual process of design, or is inappropriate to the particular stage of design (ie early, late). Hence, design elicitation procedures should allow the expert to problem-solve in a natural way (eg. Protocol Analysis of design situations), be flexible (eg. the flexible personal interview), and incorporate various methods that cover a diversity of teaching/learning styles (ie. adopting a global approach to knowledge elicitation, with an emphasis on human-human communication in order to learn and develop a shared understanding).

Secondly, care should be taken to address the medium by which designers communicate. Therefore, the use of diagrammatical methods appear an appropriate medium of communication with designers (eg Landsdown, 1987). For example, Li and Bourne (1989) argue that diagrammatical, qualitative networks are useful in the development of inspectable models of design.

It can be argued that science is not what science is perceived to be, ie that scientists do not in practice necessarily follow what is regarded as (theoretic) scientific method. However, the problem facing design knowledge acquisition is the failure to recognise the 'less appealing' styles of cognition, that have predominated traditional KBS applications. For example, seldom are "Impulsive" thoughts (Kagan, 1965) regarded as useful in categorisation or analysis problems, but they are nevertheless considered to be useful in design, particularly in the early stages. Cross-examining the expert for explanations that arise from impulsive decision making is unlikely to lead to comprehensible answers, it is more likely to result in answers of the form "I just felt like it". What it important is to understand that this type of answer is legitimate, and not to attempt to rationalise in a decompositional way.

Important psychological processes, such as visual perception, probably contribute significantly to design but are unlikely to be available to verbal access. Elicitation procedures for design must at least provide the medium by which spatial and holistic forms of knowledge can be recorded, represented and communicated. It is assumed here that some useful design information is available through verbal articulation. Considerations for the elicitation of verbal design data centre on the need to accurately record data, and in the confidence that the obtained information is accurate, and not a rationalisation of what the originator cannot feasibly introspect. The arguments regarding the validity (or invalidity) of verbal reports as a means of explaining mental behaviour are well documented (Nisbett and Wilson, 1977; Bainbridge 1979, 1986; Berry and Broadbent, 1984; Ericsson and Simon 1980). Protocol analysis studies of design activity have been conducted by design researchers with the aim of making generalisations about the design process (eg Eastman, 1970; Akin, 1979), and Johnson et al (1988) discuss a knowledge elicitation case study in industrial experimental design.

In Protocol Analysis non-verbal information may also be unconsciously communicated, (ie through gesticulation). A video camera can capture much of this non-verbal communication, also recording spatial aspects of the design task. Therefore, we would reject the view that "Video ... seems too expensive and impractical to recommend" (Gammack, 1987). It is our view that the visual and spatial data makes an essential contribution to design elicitation. Indeed, the absence of visual records may render verbal protocols meaningless at times because of the use of words as indexes (eg this, that, here, and there). Such utterances can only

[5] Refer to footnote 1

be understood in conjunction with the artifact to which they refer. Anaphoric references are often used and gestures form an essential part of the communication. Asking the subject for clarification may interfere, and explanations will be suspect if the designer is required to rationalise involuntary behaviours. Video records can capture the holistic, diagrammatic, and dynamic nature of the task, and in addition there is the soundtrack. An interesting observation on design tasks that include natural drawings is that the process of design is often traceable. For example, it is easier to trace the development of an idea from conception to solution in design than it might be in other domains of expertise (eg medical diagnosis), where often only the results are externalised (post-decision explanations are subject to distortion and rationalisation), because in the drawings produced the designer has a vivid reminder of available alternatives and decisions.

Ward (1984) argues that designers adopt specialised representations and procedures, to acquire and communicate spatial information. However, the form and nature of communication is likely to change throughout the various stages (eg Christopher-Jones, 1963) of the design process. For example, at the divergent stage, communication may be predominantly spatial, with many drawings and quick sketches, whereas in later, convergent stages, the communication may be more verbal and explanatory. Care must therefore be taken to cover all likely forms of design communication, both verbal and visual.

We therefore maintain that it is essential for design elicitation techniques to incorporate natural drawing tasks for design problems that involve visual and spatial information. Subsequent to the task, diagrammatical methods appear a potentially useful, natural and convenient medium of communication with designers.

A holistic strategy for design knowledge elicitation

In this section we present a holistic[6] view to design knowledge elicitation. It has been suggested here that high-level strategic knowledge, and knowledge structures used by design experts exhibit complex interrelationships that are probably not decomposable. Thus, although

[6] The terminology used to describe what is meant by "Holistic" is imprecise, and contentious. Bradshaw and Nettleton (1983) referring to visual processing attempt to clarify the terminology:

"The term analytic properly applies to perceptual processing, rather than, for example, to the preparation of a response sequence, in the context of a segmental breakdown of a visual array or an auditory grouping into separate components, features or elements. In theory at least, such a breakdown may be performed either serially and sequentially or in parallel. In contrast, global, holistic, or gestalt processing can be viewed as a more immediate and possibly primitive mode of apprehension; it places more emphasis on the whole configuration and on the patterns of interrelationships between and otherwise independent of, the component parts. W.R. Garner (1978), without reference at all to the laterality literature, argues for two classes of stimulus properties, component and holistic. The component class he subdivides into dimensions (eg. brightness, loudness, form) and features (eg. presence of a horizontal line, an intersection and so on). The holistic class of stimulus properties he subdivides into 'simple wholes' (eg. 'blob processing' where an entity is not or cannot meaningfully be further subdivided or analysed), 'templates' (the simplest 'canonical' or idealised form of configuration) and 'configurations' (where the whole constitutes something over and above the sum of the individual component elements)."

explanations of the nature of sub-task or component level interactions may be provided by the expert designer, they are unlikely to be accurate models of the design activity. They are more likely to be inferred explanations lacking any true knowledge base. Put simply, the designer is unlikely to comprehend the design at the component interaction level. However, it is likely that holistic strategies for representing and dealing with complex interrelations are cogniscible by the designer, and some of this information may be available to (verbal) explanation mechanisms, even though details are probably not. It is postulated that high level holistic structures carry valuable information about design expertise. Consequently, reductionist philosophies of knowledge elicitation are less appropriate for design applications because design is intrinsically more dependent on a subtle, but nevertheless powerful, contribution of knowledge that is altogether more holistic in nature. Hatvany (1985) attempts to describe this type of design knowledge, but attains little success.

Scientific Issues	Design Issues
Descriptive (Understanding) Analysis, Diagnosis, Categorisation, Classification.	Prescriptive (Productive) Modelling, Representation, Manipulation, Integration, Intentional, Useful, Expedient
Numerical / Linguistic	Visual / Spatial / Diagrammatic
Decompositional Top-Down, Bottom-up.	Holistic
Prototyping, Transformation, Optimisation	Discovery of new Goals, Invention, Integration
Scientific Modes of Thinking Reflective, Convergent, Field Dependent, Serialistic	Design Modes of Thinking Impulsive and Reflective, Divergent and Convergent, Field Independent and Field Dependent, Holistic and Serialistic
Problem Focused	Solution Focused

Table 2 : Issues relating to Scientific and Design approaches to problem solving.

Computer tools have already been developed to aid both bottom-up and top-down approaches to elicitation (eg. Rajan, Motta and Eisenstadt, 1988). However, it is argued here that the elicitation of design knowledge should progress in a global way, to elicit higher cognitive concepts (but not through hierarchical decomposition), and the lower cognitive concepts (but not through component level construction). Understanding is often most opaque in the centre of our conception, paradoxically where the information is most interesting.

Our view is that the knowledge engineer should become familiar with information *as it*

becomes available from the most natural form of communication with an expert (human-human), preferably in a repetitive cyclic process to avoid fixed perspectives. Easily understood concepts may be learned first, and as more information emerges the process of organising, explaining and structuring the understanding will progress. Note that easily communicated concepts are likely to be both conceptually high and low, and mixed in natural discussion. The information emerging from the expert will almost certainly be unstructured and ill-defined since the designer will not know the extent or structure of his own knowledge and therefore asking him for high or low level concepts is rather impractical and ineffective.

Nevertheless, when the expert is communicating naturally, the knowledge engineer should be able to acquire information in a global way, narrowing the understanding gap more rapidly than choosing a single top-down (decomposition) or bottom-up (construction) strategy. This is in essence a holistic learning strategy (Pask and Scott, 1971), and learning is surely the purpose of knowledge elicitation.

Many knowledge elicitation researchers argue that the purpose of knowledge elicitation is to develop a model of understanding, which is neither simple nor easy, and computers are not yet capable of doing this on their own (eg Musen, 1989). We agree with this sentiment and add that models of design must be constructed by knowledge engineers. The engineer must learn and develop an understanding from whatever information is available, initially, or can be obtained when progressive developments drive the learning process.

It must be recognised that knowledge engineering is fundamentally an analysis process; at some stage it will be necessary to code information to be used in a Knowledge-Based system, and reductionist methods will need to be employed. However, a holistic approach to the knowledge elicitation process may yield requirements for knowledge representational formats or programming paradigms that are as yet inadequately catered for in current formats. ie the knowledge elicitation process may suggest that the rule format may not necessarily be sufficient. Choosing a reductionist approach from the outset is likely to distort the knowledge acquisition process, and miss essential forms of design knowledge.

The use of Intermediate Representations is an important strategy for the development of independent models (of cognition or competence), and we believe that Mediating Representations provide a suitable mechanism for the interactive and progressive development of models of design, using a global strategy, diagrammatical methods and human-human communication. The 'teachback' methodology (Johnson and Johnson, 1987) was extensively used in our studies (Tunnicliffe, 1990), and we would argue that it is of major significance to design knowledge elicitation procedure. The resulting information obtained from the above knowledge elicitation study is currently being utilised by another research group, to construct a page-layout design advice KBS. Diagrams have been recognised elsewhere (eg Johnson et al, 1988; Lee et al, 1990) to be a useful mechanism of design knowledge transference.

It is concluded here that automated tools that rely on reductionist approaches to the elicitation of design knowledge fall short of the ability to understand, model, and represent design knowledge. Simply, they lack the cognitive power of the human intellect. Qualitative and personal interview strategies currently offer the most promising approach to the elicitation of design knowledge. The need to capture and utilise visual and diagrammatic information is clearly important.

SUMMARY

Design theorists argue that traditional notions of scientific cognition have constrained the development of adequate design theories, and by the same token, we maintain that traditional views of knowledge elicitation have hindered the progression of design knowledge elicitation procedure. Important concepts arising from design, such as holistic and spatial metaphors are poorly considered in current views of design knowledge elicitation. Such characteristics question the appropriateness of hierarchical top-down decomposition or bottom-up construction approaches, and design can not be directly equated to analysis or classification problems. Therefore, we have argued that methods of design knowledge elicitation wholly oriented towards these conceptions are limited.

The design process is subtly different to scientific method (Table 2). The solution-focused nature of designing is in stark contrast to the problem-focused nature of scientific problem solving. For example, a large number of possible solutions to a design problem can be extensively pruned by a small piece of information, a personal preference, or an arbitrary decision. In many areas of design, design goals can not be specified, initially, and it is common for design goals to be dynamically discovered through the design process. Therefore, doubts can be raised as to the appropriativity of the Generic Task approach to specifying design expertise.

We argue that whilst engaged in design, designers can not rationalise a smooth progression from either high level abstractions through to components or from components upwards. We suggest that such approaches are neither practical, nor appropriate to design, and we maintain that a more holistic approach to design knowledge elicitation is required. Asking the designer to describe details of complex interactions in a bottom up manner will probably introduce inaccuracy in the knowledge elicitation process since the expert designer is unlikely to consider details of individual components, and it is not practical to define, or decompose into discrete parts, entities that are subject to complex interaction.

An important aspect of the holistic approach is in the use of diagrammatic and spatial communication mediums and to use appropriate recording methods. It is inappropriate to squeeze design knowledge into predefined knowledge structures, that (arguably) attain success in scientific problem solving domains, when the form and nature of design knowledge is so poorly understood. We further argue that it is premature to rely on automated techniques and metaphors of design that have little credibility with researchers of design.

It has not been established, using current approaches, that it is in fact practical to conduct design knowledge elicitation, as an equivalent to scientific knowledge elicitation. This point is reiterated by the observation that so few practical accounts of design knowledge elicitation exist, and those that are reported are in domains that seem to miss essential aspects of the design process. Indeed, it is not at all clear what methods are appropriate for design knowledge elicitation and we do not claim to have all the answers. However, it is clear that better methods are required and this discourse aimed to provoke greater thought and endeavour to be applied to the field of design knowledge elicitation.

References

Akman, V., ten Hagen, P., Rogier, J. and Veerkamp, P. (1988). Knowledge Engineering in Design. Knowledge Based Systems, (March, 1988), Vol 1, (2), pp67-77. Butterworth.

Akin, O. (1979). In Cross, N (Ed) (1984), Developments in Design Methodology. Wiley.

Alem, L., and Maher, M.L. (1991). Learning Generalised design prototypes. Proceedings of the Australian Workshop on Knowledge Acquisition for Knowledge-Based Systems, Pokolbin, August 1991, (University of Sydney), pp 17-33.

Allen, W. (1984). Root causes of degradation. In ICE, Design life of buildings. Telford.

Archer, L.B. (1963). Systematic Method for Designers. Design magazine, 176, pp52-57.

Archer, L.B. (1979). Whatever Became of design Methodology ? Design Studies, 1, 1 pp17-18.

Archer, L.B. (1991). The nature of research into Design and Design Education. Keynote address: DATER 91, 4th National Conference on Design and Technology Education Research and Curriculum Development, 5-7 September 1991. Loughborough University of Technology. pp 1-11.

Bainbridge, L. (1979). Verbal reports as evidence of the process operator's knowledge. International Journal of Man-Machine Studies, 1, pp411-436.

Bainbridge, L. (1986). Asking questions and accessing knowledge. Future Computing Systems, 1, no 2, pp143-149.

Berry, D.C. and Broadbent, D.E. (1984). On the relationship between task performance and associated verbalisable knowledge. The Quarterly Journal of Experimental Psychology, 36(a), pp209-231.

Bignell, V., Peters, P. and Pym, C. (1977). Catastrophic failures. The Open University Press.

Bignell, V. and Fortune, J. (1984). Understanding system failures. Manchester University Press.

Bijl, A. (1985). An approach to design theory. In Yoshikawa and Warman (Eds), Design theory for CAD. North-Holland. pp3-31.

Boose, J.H. and Bradshaw, J.M. (1987). Acquiring design and analysis knowledge for Knowledge-Based systems. In ten Hagen, P.J.W. and Tomiyama, T. (Eds), Intelligent CAD systems I: Theoretical and Methodological Aspects. Springer-Verlag. pp128-145.

Boose, J.H., Shema, D.B., and Bradshaw, J.M. (1990). Capturing Design Knowledge for Engineering Trade Studies. In Wielinga et al (Eds), Current Trends in Knowledge Acquisition. IOS press.

Bradshaw, J.L. and Nettleton, N.C. (1983). Human Cerebral Asymmetry. Prentice Hall.

Bradshaw, J.M. and Boose, J.H. (1990). Decision analysis techniques for knowledge acquisition: combining information and preferences using Aquinas and Axotl. International Journal of Man-Machine Studies (1990), 32, pp121-186.

BRS (1978). Building failure. BRS research series, Vol 5. The Construction Press.

Buck, P., Clarke, B. and Lloyd, G. (1991). A top-Down approach for capturing design knowledge. Proceedings of the Fifth European Knowledge Acquisition for Knowledge-Based Systems Workshop, Crieff, Scotland, May 1991. (np).

Carroll, J.M. and Rosson, M.B. (1985). Usability specifications as a tool in iterative development. In Hartson, H.R. (Ed), Advances in Human-Computer Interaction. Ablex. pp1-28.

Chandrasekaran, B. (1988). Generic Tasks as building blocks for Knowledge-Based Systems: The diagnosis and routine design examples. The Knowledge Engineering Review, Vol 3, 3 (September), pp183-210.

Christopher-Jones, J. (1963). A method of Systematic design. In Jones, J.C. and Thornley, D. (Eds), Conference on design methods. Permagon.

Cross, A. (1984). Towards an understanding of the intrinsic values of design education.

Design Studies, January 1984, Vol 5, No. 1. pp 31-39.
Cross, N., Naughton, J. and Walker, D. (1981). Design Method and Scientific Method. Design Studies. Vol 2, no 4, pp 195-201.
Cross, N. (1982). Designerly ways of knowing. Design Studies, October 1982, Vol 3, No. 4. pp 221-227.
Cross, N. (Ed) (1984). Developments in Design Methodology. Wiley, 1984.
Cross, N. and Nathenson, M. (1980). Design methods and learning methods. In Jacques R and Powell J (Eds), Design : Science : Method. Westbury House. pp 281-294.
Darke, J. (1979). The Primary Generator and the design process. Design Studies, 1, (1), pp36-44. Butterworth.
Dreyfus, H.L. and Dreyfus, S.E. (1986). Mind over machine, The power of human intuition and expertise in the era of the computer. Basil Blackwell.
Eastman, C.M. (1970). On the analysis of intuitive design processes. In Moore, G.T. (Ed), Emerging methods in environmental design and planning. MIT press.
Edwards, B. (1979). Drawing on the right side of the brain. Souvenir press, 1979.
Ericsson, K.A. and Simon, H.A. (1980). Verbal Reports as Data. Psychological Review, 87, (3), pp 215-251.
Ferguson, E.S. (1977). The mind's eye : non-verbal thought in technology. Science, Vol 26, pp 827-837.
Gammack, J.G. (1987). Different techniques and different aspects on declarative knowledge. In Kidd, A.L. (Ed), Knowledge Acquisition for Expert Systems. pp137-164.
Garner, W.R. (1978). Aspects of a stimulus: Features, dimensions and configurations. In Rosch, E. and Lloyd, B.B. (ed) (1978), Cognition and characterization. Erlbaum.
Gero, J.S., Radford, A.D., Coyne, R. and Akiner, V.T. (1985). Knowledge-Based Computer-Aided Architectural Design. In Gero, J.S. (Ed), Knowledge Engineering in Computer Aided Design. (1984, Budapest Hungary). Elsevier. pp57-88.
Gero, J.S. (1989) (Ed). Artificial Intelligence in design. Proceedings of the fourth international conference on the applications of Artificial Intelligence in Engineering. (July 1989, Cambridge, UK). Springer-Verlag.
Guena, F. and Zreik, K. (1989). An Architect Assisted Architectural Design System. In Gero, J.S. (Ed), Artificial Intelligence in design. Proceedings of the fourth international conference on the applications of Artificial Intelligence in Design. (July 1989, Cambridge, UK). Springer-Verlag. pp159-179.
Hatvany, J. (1985). An attempt at a holistic view of design. In Yoshikawa and Warman (Eds), Design Theory for CAD. North-Holland. pp131-142.
Hillier, B., Musgrove, J. and O'Sullivan, P. (1972). Knowledge and Design. In Mitchell, W.J. (Ed), Environment Design: Research and Practise. University of California.
Holt, J.E., Radcliffe, D.F. and Schoorl, D. (1985). Design or problem solving, a critical choice for the engineering profession. Design Studies, 6, pp107-110.
Johnson, L. and Johnson, N.E. (1987b). Knowledge Elicitation involving teachback interviewing. In Kidd, A.L. (Ed), Knowledge Acquisition for Expert Systems. Plenum. pp91-108.
Johnson, P.E., Zaulkerman, I, and Garber, S. (1988). Specification of Expertise. Knowledge-Based Systems, Vol 1, pp 125-145.
Kagan, J. (1965). Reflection-impulsivity and reading ability in primary grade children. Child development, 36, pp609-628.

Kamal, S.Z., Mistree, F., Sorbab, J. and VanArsdale, W.E. (1989). The development of an inductive learning system for design using experimental information. In Gero, J.S. (Ed), Artificial Intelligence in design. Proceedings of the fourth international conference on the applications of Artificial Intelligence in Design. (July 1989, Cambridge, UK). Springer-Verlag. pp521-538.

Kornell, J. (1987). Formal thought and narrative thought in knowledge acquisition. In Boose, J.H. and Gaines, B.R. (Ed) (1989), Knowledge Acquisition for Knowledge-Based Systems 1. Academic Press. pp35-44.

Landsdown, J. (1987). Computers and visualisation of design ideas: possibilities and promises. In Maver, T. and Wagter, H. (Ed), CAAD '87 Futures. Elsevier. pp71-80.

Landsdown, J. (1988). The designer's information environment : Tools for Design Knowledge manipulation. In Hadgraft, R. and Young, W. (Eds), Conference proceedings of the symposium of Knowledge Based Systems in civil engineering. Monash University, Clayton, Australia.

Landsdown, J. and Roast, C. (1987). The possibilities and problems of knowledge-based systems for design. Environment and planning B, (14), pp255-266.

Lawson, B.R. (1972). Problem solving in architectural design. Ph.D. thesis. University of Aston in Birmingham.

Lawson, B.R. (1979a). Cognitive strategies in architectural design. Ergonomics, Vol 22, 1, pp59-68.

Lawson, B.R. (1980). How designers think. Architectural press, 1980.

Lee, J.K., Lee, I.K. and Choi, H.R. (1990). Automatic rule generation by the transformation of expert's diagram: LIFT. International Journal of Man-Machine Studies, 32, pp275-292.

Li, X. and Bourne, J. (1989). Building Inspectable Models of Networks : an example in Power Distribution Systems. In Gero, J.S. (Ed), Artificial Intelligence in design, Proceedings of the fourth international conference on the applications of Artificial Intelligence in Design. (July 1989, Cambridge, UK). Springer-Verlag.

Luckman, J. (1967). An approach to the management of design. Operational Research Quarterly, 18, 4, pp343-358.

McConnell, M. (1987). Challenger: A major malfunction. Simon and Schuster.

McSherry, D (1991). Eliciting constraints and non-monotonic preferences for multi-criterion decision making. Proceedings of the Fifth European Knowledge Acquisition for Knowledge-Based Systems Workshop, Crieff, Scotland, May 1991. (np).

Magee, H.K. (1987). The elicitation of knowledge from designers. Design Studies, Vol 8, no 2. pp62-69.

Marcus, S. (1988). Understanding Subtasks from a Piecemeal Collection of Knowledge. 3rd AAAI-sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, November, 1988. no 19.

Marcus, S. and McDermott, J. (1987). SALT : A knowledge acquisition language for propose-and-revise systems. Technical report, Department of Computer Science, Carnegie-Melton University, Pittsburgh, PA.

Musen, M.A. (1989). Automated Generation of Model-Based Knowledge-Acquisition Tools. Pitman.

Nader, R. (1965). Unsafe at any speed: the designed-in dangers of the American automobile. Grossman.

Nisbett, R.E. and Wilson, T.D. (1977). Telling more than we know : verbal reports on mental processes. Psychological Review, 84, (3), pp 231-259.

Pask, G. and Scott, B. (1971). Learning and teaching strategies in a transformation skill. British journal of Mathematical and Statistical Psychology, 24, pp205-229.

Pask, G., and Scott, B. (1972). Learning strategies and Individual Competence. International Journal of Man-Machine Studies, 4, 3, pp217-253.

Quinlan, J.R. (1987). Simplifying decision trees. International Journal of Man-Machine Studies, 27, pp 221-234.

Rajan, T., Motta, E. and Eisenstadt, M. (1988). Aquist: A tool for Knowledge Acquisition. In Kelly, B. and Rector, A. (Eds), Research and Development in Expert Systems V. CUP. pp113-125.

Scott, G. (1976). Building disasters and failures, a practical report. The Construction Press.

Scrivener, S.A.R., and Palmen, H.K. (1991). An Analysis of Face-to-Face Drawing Activity. DATER 91, 4th National Conference on Design and Technology Education Research and Curriculum Development, 5-7 September 1991. Loughborough University of Technology.

Scrivener, S.A.R., Clarke, A.A., Connoly, J.H., Garner, S.W., Schappo, A., Clark, S.M., Palmen, H.K. and Smyth M.G. (1992). ROCOCO Phase 1 report. LUTCHI Research Centre, 1992.

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 Knowledge Acquisition for Knowledge Based Systems Workshop, Banff, Canada, October, 1989.

Simon H.A. (1973). The structure of ill-structured problems. Artificial Intelligence, 4, pp181-200. North-Holland.

Smithers, T., Conkie, A., Doheny, J., Logan, B., and Millington, K. (1989). Design as intelligent behaviour: an AI in design research programme. In Gero, J.S. (Ed), Artificial Intelligence in design. Proceedings of the fourth international conference on the applications of Artificial Intelligence in Design. (July 1989, Cambridge, UK). Springer-Verlag. pp293-334.

Stewart, S. (1988). Air Disasters. Arrow Books, London.

Tate, A. (1985). A review of Knowledge-Based planning techniques. In Merry, M. (Ed), Research and development in expert Systems. pp 89-112.

Thomas, J.C. and Carroll, J.M. (1979). The Psychological study of design. Design Studies 1979, Vol 1, no 1, pp5-11.

Tovey M. (1984). Designing with both halves of the brain. Design Studies, October 1984, Vol 5, no 4, pp 219-228.

Tunnicliffe, A.J. (1990). Knowledge Elicitation in Design : A case study of page layout design. Ph.D. Thesis. Loughborough University of Technology, LEICS, UK.

Ward, A. (1984). Design cosmologies and brain research. Design Studies, October 1984, Vol 5, no 4. pp 229-238.

Whyte, R.R. (ed) (1975). Engineering progress through trouble. Institution of mechanical engineers. London.

Generalised Directive Models*

Gertjan van Heijst* Peter Terpstra* Bob Wielinga*
Nigel Shadbolt†

* Social Science Informatics
Department of Psychology
University of Amsterdam
Roeterstraat 15
1018 WB Amsterdam
Netherlands
Fax: (+31)20 5256896
email: {gertjan,terpstra,wielinga}@swi.psy.uva.nl

† AI Group
Department of Psychology
Nottingham University
University Park
Nottingham NG7 2RD
United Kingdom
email: nrs@psyc.nott.ac.uk

*The research reported here was carried out in the course of the ACKnowledge project partially funded by the ESPRIT Programme of the Commission of the European Communities as project number 2567. The Partners in this project are GEC-Marconi, University of Nottingham (both UK), Cap Gemini Innovation (F), Sintef, Computas Expert Systems (both N), Telefonica (ES), and the University of Amsterdam (NL).

Abstract

Most model driven approaches to knowledge acquisition view knowledge acquisition as a two stage process: first select an abstract model of the problem solving process and then fill in the model with the required domain knowledge. In this paper we argue that often the abstract model can only be selected after a substantial amount of domain knowledge has been acquired, invalidating the two stage assumption. We present an approach that overcomes this difficulty by viewing knowledge acquisition as a cyclic process of domain knowledge elicitation and model construction. In this approach the model constructed up to a certain point is used to direct acquisition of domain knowledge, and the elicited domain knowledge is used for subsequent refinement of the model. We describe an implemented model construction method that actively supports this cyclic process. The approach is illustrated with a real life knowledge acquisition scenario in the domain of respiratory diseases.

1 Introduction

Recently it has been widely acknowledged that predefined (reusable) abstract models of the tasks that expert systems have to perform can facilitate knowledge acquisition. This is often called the model driven approach to knowledge acquisition. These abstract models have taken the form of interpretation models (Wielinga *et al.*, 1992), generic tasks (Chandrasekaran, 1987) and task specific shells and tools (Musen *et al.*, 1988, McDermott, 1989). Although these approaches emphasize different aspects of the problem solving process and have different levels of abstraction, they are based on similar assumptions (Karbach *et al.*, 1990).

Abstract models can be used as high level templates that put constraints on the domain knowledge that is required. They direct knowledge acquisition because they make explicit what kind of knowledge is used in the problem solving process and they provide a schema for structuring the knowledge base.

The model driven approach implicitly advocates a two stage model of the acquisition process: First one acquires the abstract model, e.g. select one from a predefined set, and then the model is filled with appropriate domain knowledge. It is assumed that the correct model can be selected using global features of the task that the system has to perform, e.g. derived from descriptions of the way human experts perform the task (think aloud protocols). In this paper we claim that this is not always the case. Sometimes the form of the abstract model depends on detailed characteristics of the domain knowledge. These characteristics can only be revealed through acquisition of domain knowledge. In situations like this there is a dilemma: some domain

knowledge needs to be elicited before the abstract problem solving model can be established, but at the same time directed elicitation of domain knowledge requires an abstract model. This invalidates the two stage assumption.

A second problem is related to the use of a library of predefined models. A library contains a limited set of prototypical problem solving models. There is not always a prototypical model available for a real world problem. In such a case the developer needs to adapt a prototypical model or alternatively, construct one from scratch. Adaptation of models could be supported by the explicit representation of partial models that all realise the same problem solving step in a different way. Model construction would be aided by the availability of generic model components and constraints on the way they can be combined into larger structures. A flat library representation does not have the expressive power required here.

In this paper we propose a way to handle the complications mentioned above without loosing the advantages of model driven knowledge acquisition. The approach evolved while developing KEW, a knowledge engineering workbench which actively supports the knowledge acquisition process by suggesting which knowledge should be acquired in the next step of the acquisition process and which acquisition tool should be used to do so (Shadbolt & Wielinga, 1990, van Heijst *et al.*, 1992).

To make our argument more concrete we will focus here on one particular kind of abstract models: KADS *interpretation models*. Interpretation models appear to be a good starting point for a more flexible use of abstract models because the KADS approach provides a semi formal modeling language for describing arbitrary models.

Interpretation models describe the inference structure and the task structure of a certain domain. However, in our analysis only the inference structure has been taken into account[1]. KADS inference structures consist of *knowledge sources*, representing "primitive" inference steps, and *metaclasses*, which index domain knowledge in terms of the role that it plays in the problem solving process. There is a fixed set of knowledge sources with a predefined semantics. Although we restrict our discussion to interpretation models, we think the points we make can be adapted to other model driven enterprises.

This paper is organised in the following way: In the next section we introduce the notion of *Generalised Directive Models* (GDM's), and we describe how they can be used to avoid the problems mentioned above. Next we describe the technical details of our approach. As an illustration we will give an example of the way GDM's are used in a real life knowledge acquisition scenario. Finally we will discuss how the

[1]In fact we have implemented a method for developing a task structure using the inference structures as input (Terpstra *et al.*, 1992)

GDM's are related to other approaches to knowledge acquisition.

2 Generalised Directive Models

The problems of model selection, the library components and their representation that we mentioned above can be solved through the introduction of what we have called generalised directive models. The term "directive" is used to make explicit that the models are used to direct knowledge acquisition.

GDM's are interpretation models which leave parts of the problem solving process underspecified. We model this through the introduction of a third building block for models: the *generalised knowledge source.* A generalised knowledge source describes or subsumes a non primitive problem solving step that represents the fact that we do not really know how the corresponding part of the problem solving process should be modeled. Another way of saying this is that a generalised knowledge source describes a set of model components with similar input/output relations. These model components could be primitive knowledge sources but they could also be partial models that consist of multiple generalised and primitive knowledge sources and intermediate metaclasses. However, these extensions do have in common that they describe the same relation between the input and the output of the generalised knowledge source.

The key idea is that these GDM's contain enough information to guide the elicitation of at least some domain knowledge. At the same time, the elicited domain knowledge will reveal new information about the domain, and this new information can be used to further refine the underspecified parts of the GDM.

The use of GDM's is based on three related principles:

- Knowledge Acquisition is a Cyclic Process:

  We view knowledge engineering as an iterative process of elicitation of domain knowledge, integration of the elicited knowledge with previously acquired knowledge and evaluation of the acquired knowledge to assess the current state of the acquisition process. *Model construction is an integral part of this cycle.*

- Compositionality:

  A directive model describes a relation between inputs and outputs. With the extension of a (partial) model we mean the set of input/output combinations for which that relation holds. The compositionality principle states that the

extension of a model only depends on the extensions of its parts and the way they are related. This implies that *model parts with a similar extension can be exchanged, leaving the extension of the model as a whole unaffected.*

- A Delayed Commitment Strategy:

  The third principle states that knowledge engineers should only commit themselves to a particular model if there is sufficient evidence that it is the "right" model. This reflects the idea that backtracking on abstract model construction is difficult and should be avoided when possible[2].

Generalised directive models enable us to interleave the two stages of the knowledge acquisition process. It is still required that there is an initial abstract model of the problem solving process before the domain knowledge is elicited. However, this initial GDM may be extremely general, leaving unspecified all parts that depend on properties of the domain knowledge. As we shall see this model can then be used to instigate acquisition of domain knowledge.

We can exploit the semantics of knowledge sources to determine the domain knowledge required to instantiate them. For example, if the model contains a primitive knowledge source "taxonomic abstraction", a domain taxonomy will be required. The generalised knowledge sources can direct knowledge acquisition in another way. Generalised knowledge sources have a limited set of possible extensions, whose applicability depends on features of the domain knowledge. Generalised knowledge sources direct knowledge acquisition by focusing on eliciting domain knowledge that reveals these features.

In the next cycle of the acquisition process the newly elicited knowledge can be used to refine the GDM, by filtering the possible extensions of the generalised knowledge sources. If only one possible extension is left, it takes the place of the generalised knowledge source.

3 The GDM grammar

In KEW the library of GDM's is represented as a context sensitive generative grammar. By analogy with natural language we could say that the final directive models are sentences, the generalised knowledge sources are intermediate constituents, and

[2]The problem of backtracking or *rolling back* a model is a substantial one which we do not have the space to pursue here. At the very least it means that the problem solving roles assigned to domain knowledge have to be changed or modified.

```
-------------------------------------------------------------------------
?input1, NT-Abstract, ?output --> (?input1, Domain-Taxonomy),
                                  T-Taxonomic-Abstraction,
                                  ?output.

Conditions: there is a domain taxonomy that relates objects in
            the metaclass unified with variable ?input1 through
            taxonomic (isa) relations with objects unified with
            variable ?output.
-------------------------------------------------------------------------
```

Figure 1: example of a GDM rewrite rule

the knowledge sources and metaclasses are words. The model construction steps in each acquisition cycle correspond to the application of rewrite rules. Each of the rewrite rules has associated conditions that have to be true before the rule may be applied. These conditions are the link between the abstract model and the features of the domain knowledge.

Let us use *abstraction* as an example. In a particular GDM grammar a generalised knowledge source such as the non terminal symbol `NT-Abstract` can be rewritten in several ways. One of the candidates is the terminal symbol `T-Taxonomic-Abstraction`. The corresponding rewrite rule is given in figure 1. The symbols `?input1` and `?output` are variables that can be unified with metaclasses. The variable scopes are limited to the containing rewrite rule and its associated conditions. The given rule reflects the following idea: If there is a generalised knowledge source `NT-Abstract` in the GDM with input `?input` and output `?output`, then this generalised knowledge source may be replaced with the knowledge source `T-Taxonomic-Abstraction` with as input metaclasses `?input` and `Domain-Taxonomy` and with output `?output`. The metaclasses `?input1` and `?output` already exist in the model before the rule is applied. The metaclass `domain-taxonomy` is introduced in the model when the rewrite rule is applied. The rule may only be applied if the associated conditions are satisfied. In this particular example there is only one condition: The existence of a domain taxonomy that contains taxonomic links between the inputs and the outputs. In figure 1 and the scenario in the next section we describe the conditions in natural language. In the our implementation, where the system checks whether the conditions are satisfied, we use a simple condition-description language.

The use of input and output variables in the left hand sides (*LHS*) of rewrite rules

is not conventional in grammars, so it needs further explanation. There are two reasons for the introduction of these variables. The first reason is that the *RHS* of a rewrite rule does not necessarily have a strict linear internal structure. A metaclass associated with the non terminal in the *LHS* of the rule can be associated with more than one knowledge source and non terminal in the *RHS* of the rule. To express such facts in the rewrite rules we need labels for these metaclasses. The variables are a way of labeling the metaclasses. The second reason for the introduction of input and output variables is that we need the ability to refer to the input and output metaclasses in the conditions.

The grammar is also extended with special model variables. These variables can be used for model wide communication about the existence of particular types of knowledge in the domain. They function as stores for information that might be needed at different places in the model. Metaphorically, these variables are the GDM equivalent to anaphora in natural language. For example, if the same rule is applied twice in the construction history of a GDM, and in the *RHS* of this rule there is a reference to a causal model of the domain, it would be more appropriate to refer to the same metaclass in the GDM than to a different but similar metaclass. This can be done by storing the causal model in a special variable `?causal-model`, that can be read by other rewrite rules. We consider these model wide variables as a concession: formal elegance is sacrificed for expressive power. For this reason we have limited the use of these variables to "features of the domain that are independent of the problem solving process", like the existence of a structural model of the object that is reasoned about, or the universe of observables.

Currently we have implemented a grammar for diagnostic problem solving. Although our current grammar is small and implementing a grammar that can handle a wider range of problem types remains a major enterprise, we consider our results as promising. The rules in our grammar are intuitively sensible and they seem to produce appropriate models. For example, our grammar is able to generate the "classical" models for heuristic classification (Clancey, 1985), cover and differentiate (Eshelman, 1988) and systematic diagnosis (Wielinga *et al.*, 1992), together with a wide range of less well known models that seem intuitively sensible. We have not been very specific about the nature of the conditions on the rewrite rules. The conditions are the interface between the grammar formalism and the empirical knowledge about problem solving processes. The conditions that we use in the example are established empirically, through a careful analysis of existing knowledge acquisition scenario's. Because we were mainly interested in the conceptual modeling phase of knowledge acquisition our conditions mainly have to do with "knowledge availability". However, we expect that *design* oriented conditions could be incorporated, reflecting the computational constraints of the target system.

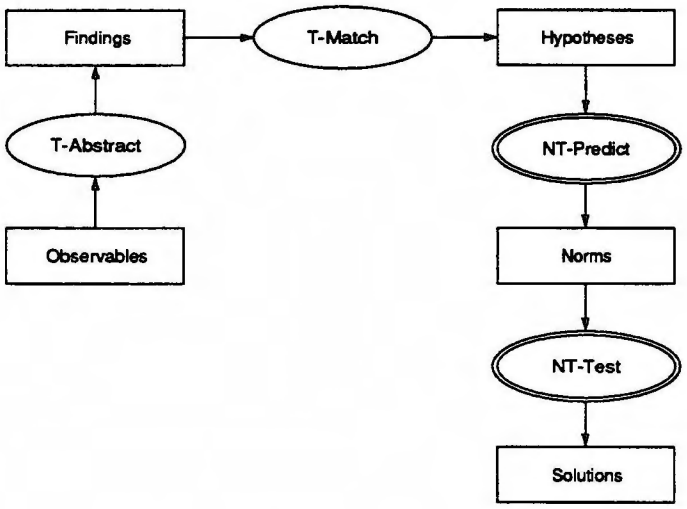


Figure 2: the current GDM in the scenario

4 A fragment of a GDM scenario

In this section we will illustrate how a GDM grammar directs the knowledge acquisition process. The example is taken from a knowledge acquisition scenario to build a system for diagnosis of respiratory diseases (Terpstra *et al.*, 1992).

At a certain moment in the scenario we have as GDM the model that is shown in figure 2. The boxes in the figure represent metaclasses, the ellipses represent primitive knowledge sources, and the double ellipses represent generalised knowledge sources. This model is generated through a cyclic process of knowledge elicitation and model refinement. In the acquisition process so far the knowledge engineer has concentrated on the left side of the model, which is the part where hypotheses are generated. This part of the model is completely refined and contains only primitive knowledge sources, and most of the domain knowledge that is required to execute this part of the problem solving process is already elicited. In particular, most diseases (hypotheses in the model) are identified. The hypothesis generation part of the model is similar to the hypothesis generation part in the classical model for heuristic classification (Clancey, 1985). However, the part of the model where the hypotheses are tested is still underspecified. We know that hypotheses are tested by predicting an observable manifestation (a `norm`) that is expected when the hypothesis is true, but how this prediction is done is still unclear. For this reason the knowledge engineer decides to concentrate on the prediction step in the model.

The GDM grammar contains four rules to rewrite the non terminal symbol

```
?input1,NT-Predict,?output --> (?input1,?causal-model,Specification-Rules)     [1]
                               T-specify,
                               ?output.

        conditions:  there is a causal model in the domain that can be
                     used to derive observable effects of deficiencies
                     in the the system that is reasoned about.

?input1,NT-Predict,?output --> (?input1,Hypothesis-Norms),                    [2]
                               T-Select,
                               ?output.

        conditions:  Every hypothesis in ?input1 has an associated norm
                     that must be true if the hypothesis is true.

?input1,NT-Predict,?output --> (?input1,Refinement-Rules),                    [3]
                               T-refine,
                               Refined-Hypotheses,
                               NT-Predict,
                               ?output.

        conditions:  Not every hypothesis in ?input1 has an associated norm,
                     but every hypothesis in ?input1 can be refined to
                     another hypothesis with an associated norm that must
                     be true if the refined hypothesis is true.

?input1,NT-Predict,?output --> (?input1,Generalisation-Rules),                [4]
                               T-Generalise,
                               Generalised-Hypotheses,
                               NT-Predict,
                               ?output.

        conditions:  Not every hypothesis in ?input1 has an associated norm,
                     but every hypothesis in ?input1 can be generalised to
                     another hypothesis with an associated norm that must
                     be true if the generalised hypothesis is true.
```

Figure 3: rewrite rules for non terminal symbol NT-Predict

NT-Predict. These rules are shown in figure 3. The main goal of the knowledge engineer at this stage in the acquisition process is to determine which of the four rules should be applied. As can be seen in the conditions this decision depends on detailed characteristics of the application domain. The first rule is only applicable if there is a causal model of the respiratory system that can be used to derive the observable manifestations of a certain respiratory disease. Because of this the knowledge engineer starts looking for such a causal model. However, in the think aloud protocols he finds evidence that many diseases have unique identifiers, and that no causal model is used to predict the norms for hypotheses. This rules out the first rule in figure 3. Since many diseases have unique identifiers the knowledge engineer suspects that the second rule is the correct rule. For this reason he tries to elicit the associated norms for the possible hypotheses. Doing this the knowledge engineer discovers that not every possible hypothesis has an obvious associated norm that can be used to decide if the hypothesis is true. This proves that rule 2 is not the correct rule either. However, additional KA reveals that every hypothesis can be refined to a more specific hypothesis that has an associated norm. At this stage the knowledge engineer knows that the third rule for NT-Predict is applicable, because the condition for that rule is satisfied. Just applying rule 3 is not sufficient. The recursive nature of the rule means that after its application the directive model still contains the non terminal NT-Predict. However, now the *refined* hypotheses are the input for NT-Predict and we know that all the refined hypotheses have associated norms so rule 2 can be applied directly after rule 3.

It is worth mentioning that application of both rule 4 and rule 3 would have caused our system to mimic the behaviour of NEOMYCIN (Clancey & Letsinger, 1984). NEOMYCIN evaluates hypotheses by first testing a generalised form of the hypothesis under consideration and then subsequently refine this hypothesis.

The fragment clearly demonstrates that sometimes a detailed analysis of domain knowledge is required to reveal the correct form of the abstract model: the question whether every generated hypothesis has an associated norm can hardly be considered as a global characteristic of the task. The fragment is also a good illustration of the delayed commitment strategy. The third grammar rule for NT-Predict is only applied when the knowledge engineer has found sufficient evidence that that rule does indeed capture the inferences made by the domain expert.

5 Relation to other approaches

Although the usefulness of model driven knowledge acquisition is now generally acknowledged, there exist several approaches to the problem of what the nature of

such models should be and how precisely models should be used in knowledge acquisition. The approach taken in Protege I and II (Musen, 1989, Puerta *et al.*, 1991) starts from the view that a task-specific model of the application is necessary to support the acquisition of the static domain knowledge. Such models are derived from templates (or built from scratch) through interaction with the knowledge engineer, while the acquisition of the static domain knowledge is done in interaction with the domain expert. The tools that were developed by McDermott and co-workers (McDermott, 1989, Marcus & McDermott, 1989), are also based on models for specific tasks. The MOLE system for example (Eshelman, 1988) uses a model of the *Cover & Differentiate* method for diagnostic reasoning to acquire domain knowledge from an expert. These approaches are powerful in the sense that knowledge acquisition with the expert is strongly focussed, but suffer from the severe limitation that the approach and the supporting tools are entirely task specific and thus limited in scope. Moreover, the task specificity puts a strong emphasis on task identification at an early stage in the acquisition process.

The use of KADS interpretation models (Wielinga *et al.*, 1992) is more flexible. Models can be selected for a wide variety of tasks. However, the models in KADS are less explicit with respect to the required domain knowledge than the models in OPAL. Moreover, KADS interpretation models generally require substantial modification for a specific application and this in turn requires information about the nature of the static domain knowledge. KADS does not offer any support for model modification.

The GDM approach combines the advantages of the approaches mentioned above. Because of the dynamic model construction process the GDM approach offers the strong guidance of task specific models while it still covers a wide range of tasks. Moreover, the delayed commitment strategy allows incremental task identification.

More recent developments based on KADS and the "Components of Expertise" approach (Steels, 1990, Vanwelkenhuysen & Rademakers, 1990) introduce the notion of *problem solving methods* (Benjamins *et al.*, 1992, Wielinga *et al.*, 1991) as a way of generating task-specific models. A problem solving method describes how a particular goal can be decomposed in primitive and/or non-primitive subgoals. Application of one or more methods to a goal results in a full model of the task. In addition to the knowledge about goals and how to decompose them, requirements for the domain knowledge are associated with each method. While these requirements are in general used for determining the applicability of methods, they could also be used for knowledge acquisition. Methods can thus be viewed as similar to the rewrite rules in a GDM grammar.

6 Future Research Directions

Currently we are elaborating the notion of primitive knowledge sources. Following the KADS ideas, we have assumed that there was a particular set of canonical inferences that could be considered primitive. We use this assumption to have a stopping criterion for our generative grammar. But what makes one inference primitive and another non primitive? Or, in GDM terminology, why is a particular knowledge source a terminal symbol or a non terminal symbol of the grammar?

We think that this question can be answered using Clancey's notion of a *Situation Specific Model* (SSM) (Clancey, 1992). The SSM is a model of the current state of the solution for the problem at hand. The primitive inferences, that is, the inferences in the final GDM, are operators that can modify the SSM. Because of this, the granularity of the GDM determines which operations are possible on the SSM, and thus the expressiveness of the SSM. Since the SSM is the basis for the explanation facility the GDM also determines the explanatory power of the target system. Thus, using more coarse primitive inferences leads to decreasing explanatory power. This argument could also be used in the other direction: the requirements for the explanation facility determine the grainsize of the final GDM and thus, which inferences should be considered primitive.

We have described a method for knowledge acquisition that differs from earlier approaches because the development of the model and the acquisition of static domain knowledge are interwoven. Initially a highly abstract model of the reasoning task is used to acquire basic domain knowledge. Subsequently the model is refined using properties of the already acquired domain knowledge. This approach allows a much more flexible and iterative knowledge acquisition process than the two-stage process used in many approaches. Although our grammars only contain a limited number of rules we are confident that at least for well understood tasks such as diagnosis a full grammar can be defined. Such a complete GDM would represent a generative theory of problem solving models.

We have implemented a number of tools within the Knowledge Engineering Workbench (KEW) to support the construction of GDMS and the operationalisation of their contents (Terpstra *et al.*, 1992). In particular, we have developed tools which

- enable the user to graphically construct GDMS
- interview the user of KEW about domain features to help resolve the conflict set of applicable GDM rewrite rules
- apply GDM rules to update the abstract model of problem solving

- decompose the current model into individual KA tasks to perform
- differentially selects KA tools with which to elicit application knowledge for various model components
- produce a task layer interpretation for the current GDM
- run domain layer knowledge using the GDM model as a meta-level control structure.

Whilst we do not pretend to have definitive GDM grammars or accounts of the knowledge required to select between competing models and KA methods, we do believe that we have developed a powerful infrastructure. This infrastructure provides us with a means of researching further the structures required to effectively carry out model driven acquisition.

References

Benjamins, V., Abu-Hanna, A., & Jansweijer, W. (1992). Dynamic method selection in diagnostic reasoning. In *Proceedings of the 12th Int. conf. on Artificial Intelligence, Expert Systems, Natural Language, Avignon'92*, Avignon. EC2. SKBS/A2/92-1.

Chandrasekaran, B. (1987). Towards a functional architecture for intelligence based on generic information processing tasks. In *Proceedings of the 10th IJCAI*, pages 1183–1192, Milano.

Clancey, W. (1985). Heuristic classification. *Artificial Intelligence*, 27:289–350.

Clancey, W. (1992). Model construction operators. *Artificial Intelligence*, 53:1–115.

Clancey, W. & Letsinger, R. (1984). NEOMYCIN: Reconfiguring a rulebased expert system for application to teaching. In Clancey, W. & Shortliffe, E., editors, *Readings in Medical Artificial Intelligence: the first decade*, pages 361–381. Addison-Wesley, Reading.

Eshelman, L. (1988). MOLE: A knowledge-acquisition tool for cover-and-differentiate systems. In Marcus, S., editor, *Automating Knowledge Acquisition for Expert Systems*, pages 37–80. Kluwer Academic Publishers, The Netherlands.

Karbach, W., Linster, M., & Voß, A. (1990). Model-based approaches: One label - one idea? In Wielinga, B., Boose, J., Gaines, B., Schreiber, G., & van Someren, M., editors, *Current Trends in Knowledge Acquisition*, pages 173–189. IOS Press, Amsterdam.

Marcus, S. & McDermott, J. (1989). SALT: A knowledge acquisition language for propose-and-revise systems. *Artificial Intelligence*, 39(1):1–38.

McDermott, J. (1989). Preliminary steps towards a taxonomy of problem-solving methods. In Marcus, S., editor, *Automating Knowledge Acquisition for Expert Systems*, pages 225–255. Kluwer Academic Publishers, The Netherlands.

Musen, M. (1989). *Automated Generation of Model-Based Knowledge-Acquisition Tools.* Pitman, London. Research Notes in Artificial Intelligence.

Musen, M., Fagan, L., Combs, D., & Shortliffe, E. (1988). Use of a domain model to drive an interactive knowledge editing tool. In Boose, J. & Gaines, B., editors, *Knowledge-Based Systems, Volume 2: Knowledge Acquisition Tools for Expert Systems*, pages 257–273, London. Academic Press.

Puerta, A., Egar, J., Tu, S., & Musen, M. (1991). A multiple-method knowledge acquisition shell for the automatic generation of knowledge acquisition tools. Technical Report KSL-91-24, Stanford University, Knowledge Systems Laboratory, Medical Computer Science, Stanford University, Stanford, California, 94305-5479.

Shadbolt, N. & Wielinga, B. (1990). Knowledge based knowledge acquisition: the next generation of support tools. In Wielinga, B. J., Boose, J., Gaines, B., Schreiber, G., & van Someren, M., editors, *Current Trends in Knowledge Acquisition*, pages 313–338, Amsterdam. IOS Press.

Steels, L. (1990). Components of expertise. *AI Magazine.* Also as: AI Memo 88-16, AI Lab, Free University of Brussels.

Terpstra, P., van Heijst, G., Nordbo, I., Ramparany, F., Reichgelt, H., Shadbolt, N., & Wielinga, B. (1992). Ka process support in KEW. ESPRIT Project P 2576 Deliverable ACK-UVA-A2-DEL-002-A, University of Amsterdam.

van Heijst, G., Terpstra, P., Wielinga, B., & Shadbolt, N. (1992). Using generalised directive models in knowledge acquisition. In Wetter, T., Althoff, K., Boose, J., Gaines, B., Linster, M., & Schmalhofer, F., editors, *Current Developments in Knowledge Acquisition: EKAW-92*, Berlin/Heidelberg. Springer-Verlag.

Vanwelkenhuysen, J. & Rademakers, P. (1990). Mapping knowledge-level analysis onto a computational framework. In Aiello, L., editor, *Proceedings ECAI–90, Stockholm*, pages 681–686, London. Pitman.

Wielinga, B., van de Velde, W., Schreiber, G., & Akkermans, H., editors (1991). *Towards a Unification of Knowledge Modelling Approaches.* Technical Report ESPRIT Project P5248, KADS-II/T1.1/TR/UvA/004/2.0. University of Amsterdam, Free University of Brussels & Netherlands Energy Research Foundation ECN.

Wielinga, B. J., Schreiber, A. T., & Breuker, J. A. (1992). KADS: A mod-

elling approach to knowledge engineering. *Knowledge Acquisition.* Special issue on KADS, forthcoming. Also as: Technical Report ESPRIT Project P5248 KADS-II/T1.1/PP/UvA/008/2.0, University of Amsterdam.

The CommonKADS Framework for Knowledge Modelling*

Bob Wielinga † Walter Van de Velde § Guus Schreiber †
Hans Akkermans ¶

† University of Amsterdam, Social Science Informatics
Roetersstraat 15, NL-1018 WB Amsterdam, The Netherlands

§ Free University of Brussels, AI Lab
Lentestraat 50-54, B-1050 Brussels, Belgium

¶ Netherlands Energy Research Foundation ECN
P.O. Box 1, 1755 ZG Petten, The Netherlands

Abstract

In this article we present a framework for modelling reasoning processes in knowledge based systems. The aim of the framework is to integrate different lines of research and in particular, though not exclusively, the KADS approach and the Components of Expertise framework. We are especially concerned with enhanced facilities for domain modelling and with the notion of problem solving method. The resulting modelling framework, called the CommonKADS modelling framework, fits into a comprehensive methodology, called CommonKADS, that intends to cover all aspects of knowledge based applications. In this article we first present a set of principles on which our modelling framework in founded. We then describe the modelling framework itself, illustrating it with an example.

1 Introduction

There is little doubt about what expert systems do: they solve problems. However, what problem solving is, how we should describe it and whether there exists any systematicity in expert problem solving are questions that are still very much open. Finding the answers

*The research reported here was carried out in the course of the KADS-II project. This project is partially funded by the ESPRIT Programme of the Commission of the European Communities as project number 5248. The partners in this project are Cap Gemini Innovation (F), Cap Gemini Logic (S), Netherlands Energy Research Foundation ECN (NL), ENTEL SA (ESP), IBM France (F), Lloyd's Register (UK), Swedisch Institute of Computer Science (S), Siemens AG (D), Touche Ross MC (UK), University of Amsterdam (NL) and Free University of Brussels (B).

This paper reflects the opinions of the authors and not necessarily those of the the consortium.

to these questions is important since a coherent framework for the description of problem solving will allow us to compare different expert systems, increase their quality and may lead to easier communication and re-use of problem solving methods and knowledge bases. Several researchers in expert systems are converging toward a common viewpoint on these issues. This view is emerging from various interrelated lines of research on problem solving, knowledge and its representation, and expert systems.

An important line of work concerns the representation of problem solving behaviors and the characterization of commonalities among them. This starts from an analysis in terms of explicit problem solving structures. It is then observed that commonalities exist across applications in the way that problem solving behavior is realized. This leads to the notion of generic task: a class of tasks for which a certain generic problem solving mechanism and representation scheme is appropriate [Breuker & Wielinga, 1989; Chandrasekaran, 1988; Clancey, 1985b; McDermott, 1989; Musen, 1989; Steels, 1990].

Newell's concept of knowledge level is another major source of ideas [Newell, 1982]. The knowledge level was originally introduced in an attempt to resolve some confusions in the usage of the terms 'knowledge' and 'representation' in Artificial Intelligence. Basically it provides for a means to describe problem solving behavior independent of how it is realized at the symbol (representation) level. Much of current research on models of problem solving is viewed, somewhat in retrospect, as 'operationalizing' Newell's ideas for practical purposes [van de Velde, 1991].

A third important influence comes from the usage of richer representations within expert systems [Steels, 1984]. Pure rule representation is considered to be no longer sufficient, neither for the purpose of system construction nor for that of knowledge acquisition. The importance of representing domain knowledge in a more adequate way is emphasized by many researchers [Lenat & Guha, 1990; Steels, 1990]. This requires stronger modelling facilities than simple attribute-value lists, if-then rules or plain logic [Genesereth & Fikes, 1992; Gruber, 1990].

The aim of this paper is to provide a synthesis of some of these ideas. Starting from Newell's knowledge level work [Newell, 1982] and some interpretations of it [Schreiber *et al.*, 1991; van de Velde, 1991], a general framework is developed through a synthesis of the KADS 4-layer model of expertise [Breuker & Wielinga, 1989] with aspects of the Components of Expertise [Steels, 1990] and work on domain modelling (e.g., [Lenat & Guha, 1990]). The result of the synthesis is called the "CommonKADS" modelling framework. This framework is aimed at the construction of knowledge level models, or *expertise models*, and fits into a comprehensive methodology, called CommonKADS, that covers all aspects of knowledge based applications.

In the following we first formulate the principles on which the CommonKADS modelling framework is based (Sec. 2). This sets the scene for outlining the modelling framework in terms of knowledge categories, their structure and the relations between them, and the knowledge level architecture that is the basis for CommonKADS expertise models (Sec. 3). Subsequently, we give details about the individual components of expertise models and suggest constructs for a modelling language (Sec. 4). The paper ends by making some final remarks and by indicating further work.

2 Principles of the CommonKADS Modelling Approach

In this section we formulate the basic principles and hypotheses that are embodied in the CommonKADS modelling framework. Some of these principles are well known and generally accepted, while others are new. They capture a number of valuable basic insights in knowledge engineering that have guided this research.

2.1 Knowledge Application Principle

There are many possible views on what expert problem solving is and how it comes about. Becoming expert in a certain field clearly involves finding and consolidating ways to achieve one's goals in spite of the many epistemic and pragmatic problems posed by that field. Steels [Steels, 1990] views the overcoming of these problems as a major hallmark of expertise. We postulate that, for the purpose of knowledge-based systems (KBS), expertise can be expressed with the help of the knowledge application principle:

> *Effective real-world problem solving is viewed as the rational (or at least, rationalizable) application by an agent of appropriate domain- and task-specific knowledge.*

The knowledge application principle seems to be widely (albeit often tacitly) shared by the community of knowledge engineers and KBS developers, although it implies a rather specific choice in the broader context of artificial intelligence, cognitive science, and epistemology.

2.2 Modelling Principle

Knowledge engineering has traditionally been viewed as a process of extracting knowledge from a human expert, and transferring it in computational form to the machine [Boose, 1985; Diederich *et al.*, 1987]. This conventional view was also reflected in the popularity of rapid-prototyping using rule-based shells. In contrast today knowledge engineering is approached as a modelling activity: the heart of the work of the knowledge engineer lies in the actual *construction* of models.

> *The development of a knowledge-based system is seen as the construction of a set of models of problem-solving behavior, seen in its concrete organizational and application context. A KBS is a computational realization associated with these models.*
>
> *A central model in the CommonKADS methodology is the expertise model, which models the problem solving behavior of an agent in terms of knowledge that is being applied in carrying out a certain task.*

Fig. 1 summarizes the suite of models involved in the CommonKADS methodology (see [de Hoog *et al.*, 1991] for a comprehensive description of the models and the dependencies between them). The expertise model is only one of many models that are relevant in KBS development. Other models capture relevant aspects of reality such as the task that an application supports, the organizational environment within which it takes place,

the assignment of tasks to agents, their capabilities and communication, and the computational design of the KBS. It is noted that these models are engineering-type models and serve engineering purposes. An important additional point is that in the CommonKADS methodology the models are considered not only as 'steps along the way', but as independent products in their own right that play an important role during the entire KBS life-cycle.

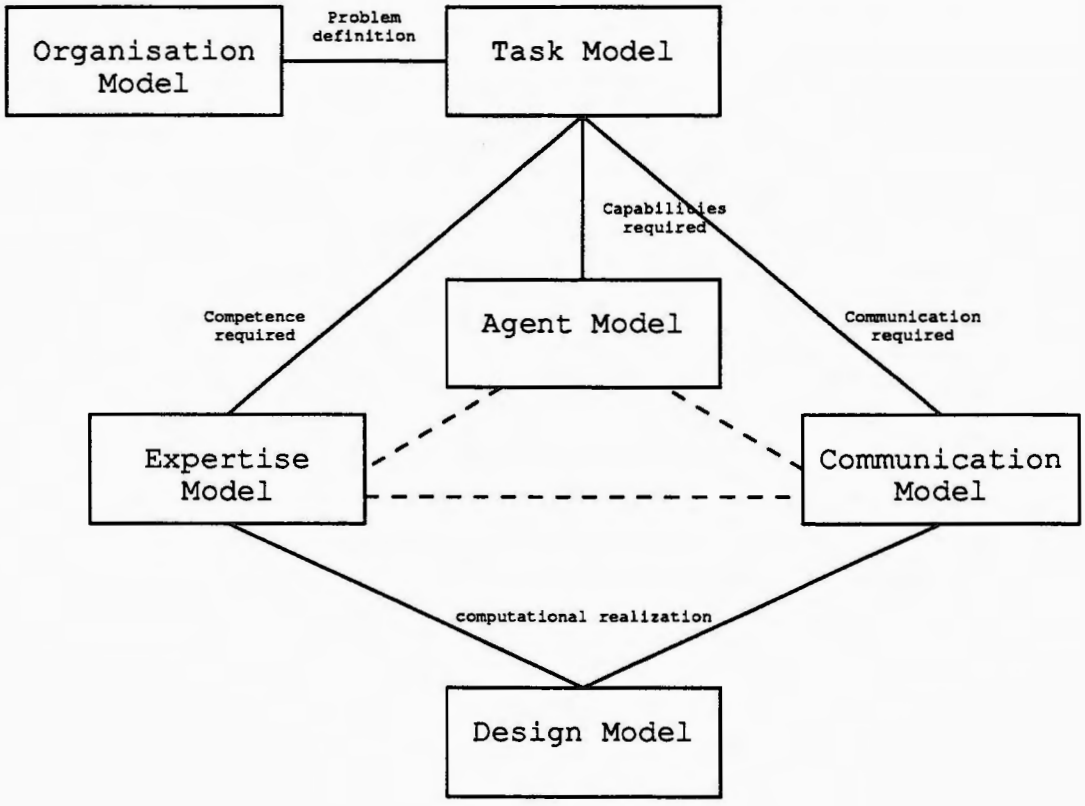


FIGURE 1: The position of the expertise model in the CommonKADS suite of models. The organization model is the tool to analyse an organization. The task model captures the global task within the organization that deals with a certain function that needs support, including the assignment of tasks to agents. The agent models describe agent capabilities, the expertise model agent competence involved in realizing the overall task. The communication model describes the communication behavior among the agents. The design model describes the structure and mechanisms of the artifacts - usually knowledge based systems - involved in the task. The lines indicate direct dependencies between elements of the models.

An expertise model, like any of the other models, can be constructed for different purposes and at different moments in time. For example, in an initial situation a task may be performed by a human expert whose capabilities are modeled, while in a later stage a KBS may take over part of the task of the human expert. The expertise model for the KBS then reflects the required expertise of the system rather than that of the human expert. Of course, a model of human expertise may serve as an important input for modelling the expertise of a KBS, but is not necessarily identical to it.

2.3 Knowledge-level Principle

Allen Newell has formulated the knowledge level hypothesis in an attempt to resolve some confusions in the usage of the terms 'knowledge' and 'representation' [Newell, 1982]. He defined a representation as "a symbol system that encodes a body of knowledge". His knowledge level hypothesis postulates the existence of knowledge independent of its representation:

> "There exists a distinct computer systems level, lying immediately above the symbol level, which is characterized by knowledge as the medium and the principle of rationality as the law of behavior." [Newell, 1982; p.99]

At this so called *knowledge level* an agent is described as having a set of goals, a set of possible actions, and a body of knowledge. Its behavior is then predicted by the so called *principle of rationality*: "If an agent has knowledge that one of its actions will lead to one of its goals, then the agent will select that action" [Newell, 1982; p.102].[1] Knowledge is characterized as a "competence-like notion, being a potential for generating action" and completely independent from representation and implementation issues.

The *knowledge level principle* as it is taken in CommonKADS, intends to give the knowledge level hypothesis practical relevance for the development of knowledge-based systems:

> *The knowledge level is the appropriate level for modelling the competence of knowledge-based systems. It calls for the description of problem solving behavior at a conceptual level that is independent from representation and implementation decisions.*

It was Clancey [Clancey, 1983] who first showed the relevance of modelling at the knowledge level for the development of knowledge-based systems. Subsequent research (see, e.g., [Akkermans *et al.*, 1992; Bylander & Chandrasekaran, 1988; McDermott, 1988; Musen, 1989; Schreiber *et al.*, 1991; Steels, 1990; Wielinga *et al.*, 1992]) has elaborated this point of view. In recent approaches to knowledge engineering the knowledge level idea is a recurring theme, though it is seldom being applied in its original form. Especially the separation of the conceptual, knowledge-oriented, aspects from representational and computational issues is seen as an important structuring and simplifying principle in KBS project management.

2.4 Role-limiting Principle

The knowledge level hypothesis does not give much guidance regarding the question *how* an intelligent agent can be described in knowledge level terms. Newell's notions of goal, action, knowledge and rationality are too coarse-grained for practical knowledge level modelling. In order to resolve this, an extension of the knowledge level principle is needed. Accordingly, an agent is thought of as imposing a structure on its knowledge by ascribing a particular role to the different components. This structure reflects the way in which an

[1] The principle of rationality is usually not considered to be part of the knowledge level model but rather act as the 'interpreter' for it.

agent frames the problem it is faced with. Acting rationally within that structure leads to a class of behaviors which adequately deal with the pragmatic and epistemological problems that the agent is faced with, such as time and resource limitations or knowledge that is incomplete or inconsistent. This idea is reflected in the role-limiting principle:

> *An intelligent agent which is faced with a particular task can be modeled as imposing on its knowledge a structure, the parts of which play different, specialized and restricted roles in the totality of the problem-solving process.*

Role limitations (a term introduced by McDermott [McDermott, 1988]) of knowledge are a basic organizing principle for modelling at the knowledge level. They delimit the class of problem-solving behaviors to those which are likely to comply with the pragmatic and epistemological characteristics of the task environment. In MYCIN for example, knowledge is configured according to the well-known heuristic classification scheme [Clancey, 1985a]. Some knowledge is used for data-abstraction, other for heuristic matching or for refinement. This leads to achieving the goals of the system and hence the employed knowledge structuring is rationally appropriate. Even when there would be a zillion ways to embark on a task a rational agent will employ one and stick to it as long as this is reasonable.[2]

2.5 Principle of Differentiated Rationality

In the knowledge level view, the behavior of an intelligent system is governed by the principle of rationality. It states that the system will take those actions that, according to its knowledge, lead to the achievement of its goals. The role-limiting principle, however, requires more elaborate principles of rationality. These must explain, first, why knowledge is structured in a certain way in order to comply with predetermined pragmatic and epistemic features of the task environment and, second, why certain knowledge is applied within the context of a specific knowledge structure. This differentiation of rationality as underlying *knowledge configuration* on the one hand and *knowledge application* on the other hand, has been called two-step rationality [van de Velde, 1991].

> *The principle of rationality must be specialized to account for the way in which the agent will structure its knowledge to approach a task in a situation with given pragmatic and epistemological characteristics.*
>
> *Within the limitations of a particular structuring and role assignment of knowledge, the principle of rationality must be further specialized to account for the way in which goals are achieved by applying the appropriate type of knowledge.*

The first part of the principle of differentiated rationality is concerned with modelling principles. It rationalizes the *structure* of the model of expertise. For example, the introduction into an expertise model (usually by the knowledge engineer, not by the system) of a data abstraction task can be rationalized by its capacity to reduce the size of the data space. The second part of the principle rationalizes control on knowledge application.

[2]The role-limiting principle has important implications for the computational tractability of complex knowledge-based systems. Epistemological role limitations of knowledge can be reflected in computational access limitations in knowledge-based systems [Akkermans *et al.*, 1992; Schreiber *et al.*, 1991].

It explains how the knowledge structure is filled in to obtain a case-specific model. For example MYCIN can be viewed as filling in the heuristic classification scheme to obtain a case-specific model that consistently explains the patient findings. The principle of rationality within the heuristic classification scheme may be phrased as 'the system uses its knowledge to reduce the size of the differential'.

2.6 Knowledge Typing Principle

The observation that different parts of the knowledge play different roles in the problem-solving process enables a distinction between different types of knowledge. In addition it appears that this differentiation does not need to be done in an ad hoc and application-dependent fashion, but rather can be characterized in generic terms. A commonly encountered notion in this respect is the separation between domain and control knowledge. In CommonKADS we employ a more refined categorization of knowledge.

> *A knowledge level model is usefully viewed as consisting of three different categories of knowledge: domain knowledge, task knowledge, and inference knowledge. Within these categories further generic types of knowledge can be distinguished.*

The domain knowledge consists of the domain-specific knowledge relevant to a given application. This entails a description of the conceptualization of the domain in terms of its ontology as well as of the structures of and relationships between the domain concepts.

The task knowledge captures the goals of an agent and the activities that are to be carried out in order to achieve them. It consists of the decomposition of the task into subtasks (which in turn may be further decomposed into subtasks), in conjunction with knowledge concerning the control over these subtasks.

The inference knowledge describes the use of domain knowledge. This is done via so called primitive inferences, that is, small-scale coherent theories employed to make certain basic inferences. By chaining primitive inferences together one obtains an inference structure: a road map of all inferences that are possible in the domain. This specifies the competence of the system.

Our experience suggests that the above present consolidated and stable categories of knowledge that recur in many applications. They are generic in the sense that they provide a vocabulary that can be applied to describe problem solving without being dependent on the specific details of task and domain of the application.

The above three knowledge categories (domain, task, inference) are necessary and sufficient for the description of application related knowledge. Apart from application related knowledge CommonKADS also supports description of problem solving knowledge. Problem solving knowledge is not a fourth category (as was the strategic knowledge in KADS-I) but is knowledge about a specific application, namely problem solving. Accordingly, problem solving knowledge can be itself described in terms of the above three domain-task-inference categories. Here, the domain knowledge contains problem solving methods, while the task-inference aspects are an operationalization of (differentiated) principles of rationality and captured in what we now call the strategic knowledge. As a whole, this vocabulary is a further extension of the knowledge level hypothesis and is intended to turn it into a practically useful approach towards the modelling of expertise.

2.7 Interaction Hypothesis

Having distinguished different categories of knowledge, a natural question to ask is how they are interrelated. On this matter, one finds quite diverse opinions in the literature concerning knowledge modelling. In the four-layer model of KADS-I [Wielinga & Breuker, 1986] the idea is that generic components related to different categories of knowledge can be modeled and represented as being independent of each other. Thus, they are thought of as 'orthogonal' dimensions of knowledge. In contrast, the Generic Task approach [Chandrasekaran, 1988] considers them as a 'package', in which different components may be distinguished but cannot be meaningfully separated: knowledge cannot be represented independent of its use. Accordingly, in the latter view knowledge components of different types are conceived of as strongly intertwined, while in the former they do not interact at all. Judging from practical KBS experience, it now appears that neither view is really appropriate, and is to be seen as an extreme point on a continuous scale. Hence:

> *A good knowledge modelling methodology is able to span the whole continuum from weak to strong knowledge interaction.*

There indeed seem to be varying degrees of task-domain coupling, depending on the application. Usually, task and inference knowledge presuppose a certain domain ontology or assume domain knowledge to be available in certain forms or schemata. On the other hand, these schemata often have a quite general character. For example, in the cover-and-differentiate method for diagnosis tasks as implemented in MOLE [Eshelman, 1988] it is required that the domain knowledge is present as a causal network. The interaction hypothesis implies that CommonKADS must be able to model multiple ontologies as well as to explicitly specify the forms and conditions under which they can be used in reasoning.

3 CommonKADS Modelling Framework

This section provides an overview of the CommonKADS knowledge level modelling framework. In the first subsection the CommonKADS view on knowledge-based problem solving is put forward. In Sec. 3.2 the main categories of knowledge are being discussed: domain knowledge, task knowledge and inference knowledge. Domain knowledge is covered in more detail than the two other categories here the framework differs most from KADS-I. Then, in Sec. 3.3 we introduce the basic structures within these categories and the concept of knowledge role that defines the interrelationships between them. Finally, in Sec. 3.4) we discuss the notion of *problem solving knowledge*. The knowledge level modelling ingredients that are introduced here are described in technical detail in Sec. 4.

3.1 Problem solving as modelling

In his original proposal for knowledge level modelling, Newell distinguished between *goals* and *knowledge*, and postulated the *principle of rationality* stating that the agent will choose an action which, according to its knowledge, leads to the achievement of its goals [Newell, 1982]. A goal, in this view, is a desired state of affairs in the world. From a knowledge level perspective the 'perception' of the world is through knowledge alone. A goal is therefore equivalent to a desired state of knowledge about the system(s) in the

world that problem solving is about. This is the basis of the view of problem solving as modelling:

> *Problem solving is the construction of a case-specific model of a system that summarizes the agent's understanding of the reality and should allow to "conclude" that the problem has been solved. This model is called the case model.*

The case model can be viewed as the variable part of the knowledge that an agent has about the systems it is confronted with. It is dynamically constructed during the problem solving process, based on the available knowledge and the information obtained from interacting with the world through *basic actions*, primarily transfer tasks that realize communication with other agents, say, users [de Greef & Breuker, 1992].

3.2 Categories of Knowledge

Though Newell's original notions of goal and knowledge are perhaps sufficient in principle, it is useful to impose structure on the description of the knowledge of an agent. For example, in Components of Expertise [Steels, 1990] knowledge is partitioned in domain models, tasks and problem solving methods. The KADS four-layer model categorizes knowledge as domain knowledge, inference knowledge, task knowledge and strategic knowledge [Wielinga & Breuker, 1986; Wielinga *et al.*, 1992].

The primary epistemological categories in CommonKADS are *domain knowledge*, *task knowledge* and *inference knowledge*. In order to describe an application at the knowledge level it is *necessary* to provide knowledge in each of these categories. An application system can not be described by, for example, domain knowledge alone.

3.2.1 Domain knowledge The domain knowledge expresses relevant knowledge about the systems - physical or not - that a task is about. That knowledge refers both to the specific systems that the problem solver is confronted with (i.e., the case model), and to the class of these systems (i.e., in terms of statements that, for the sake of problem solving, are assumed to hold for all systems of a certain kind).

The KADS-I modelling methodology was weak in the support it gave for modelling the domain. Basically, there were only two ways in which domain entities could be referred to: primitive representational terms (e.g., concepts) and meta-classes [Wielinga & Breuker, 1986]. CommonKADS provides a more comprehensive domain modelling methodology, based in particular on extensible ontologies and a powerful notion of domain model. Elements of this framework have been derived from other work, in particular from [Lenat & Guha, 1990; Steels, 1990].

> *An ontology for describing a particular domain contains the terms referring to the entities that must be distinguished as categories in the domain and the terms that refer to the relations that can hold between (instances of) the entities. Entities can be primitive, i.e. entities that exist and can be recognized in the domain, or can be abstracted, i.e. terms refer to abstractions of complexes of domain entities or to abstract concepts which have no direct correspondence in the domain.*

Lenat and Guha [Lenat & Guha, 1990] discuss two properties that an ontology should have: *epistemological adequacy* and *pragmatic adequacy*. An ontology is epistemologically adequate if it makes the distinctions that are required by the task environment, i.e. it allows us to talk about all the things that we need to talk about. An ontology is pragmatically adequate when it allows efficient problem solving. These criteria can be conflicting. Lenat proposes a way to solve the problem of conflicting requirements. The epistemologically adequate ontology is based on the most primitive notions of the domain. On top of this ontology a number of pragmatic abstractions are build which provide higher level constructs and allow more efficient reasoning to be performed. For example, in the domain of audio equipment an epistemologically adequate ontology could be based on the theory of electronics, i.e. by defining the workings of equipment in terms of electrical signals and components. For practical purposes (e.g. diagnosing faults in audio equipment) such a vocabulary would be too low level. A pragmatic ontology would introduce additional concepts like amplifier, volume control system, power supply etc. Domain knowledge formulated in such a pragmatically adequate ontology is more suitable for the reasoning task that has to use the domain knowledge. It thus represents a certain *viewpoint* on the domain. Such a viewpoint, is called a *domain model* [Steels, 1990] and it is particularly important in knowledge engineering. The way in which a domain is structured, as for example exemplified by the human expert, can provide important handles on the reasoning process.

> *A domain model is a coherent collection of statements about a domain that represents a particular viewpoint on the domain knowledge such that it is suitable for the problem-solving task. The domain model may therefore embody certain assumptions that are specific for the use that is made of it.*

The idea of viewpoint is reflected in the usage of a coherent vocabulary and a coherent structure to refer to the individual statements in the domain model. For example, a hierarchical system decomposition can be one such viewpoint on a device. It is characterized by terms such as 'part' and 'is subpart of' to refer to items of domain knowledge. Also, causal modelling views the world as 'causes' and 'effects' that are structured by one or more causal relations and uses a particular interpretation of what 'causation' means. Terms like 'part' and 'cause' are specified in the model ontology:

> *The model ontology of a domain model specifies the generic terms that a reasoning model may use to refer to the statements within that domain model.*

The model ontology is different from the domain ontology. It does not describe entities in the domain itself, but is a description of entities in the domain model. The model ontology is a *meta description* of the elements in the domain model. For example the term 'part' in the model ontology may correspond to 'integrated circuit' in the domain ontology. The statement 'The integrated circuit IC-35K4 is mounted in PCB-A-5236' in the domain knowledge may be referred to as 'part IC-35K4 is subpart of part PCB-A-5236' when using the model ontology of a hierarchical decomposition domain model.

In the above example we intentionally avoided the use of predicates or other formal constructs. Indeed, an ontology only describes the *vocabulary* for representing a domain, but not the actual representational structure. First order logic (or even natural language)

may be sufficient for describing the domain knowledge but for a domain model the representation should fit the use that is being made of it by a reasoner. For example a reasoner employing systematic diagnosis [Breuker *et al.*, 1987] requires a hierarchical decomposition of the system that is being diagnosed. The representational primitives that are the building blocks for these elements of the domain model are specified in a *model schema.*

> *A model schema describes the structure of the entities that are represented in a domain model. The type of schemata that are required for a certain domain model depend on the nature of the domain of application and on the reasoning processes that make use of the domain model.*

Model ontology and model schema together provide for a mechanism to refer in a generic way to domain knowledge which can itself be stated using an arbitrary domain dependent terminology and constructs. This allows for abstracting from irrelevant details and for the sharing of individual statements by different domain models. In addition it is the basis for reusability across domains.

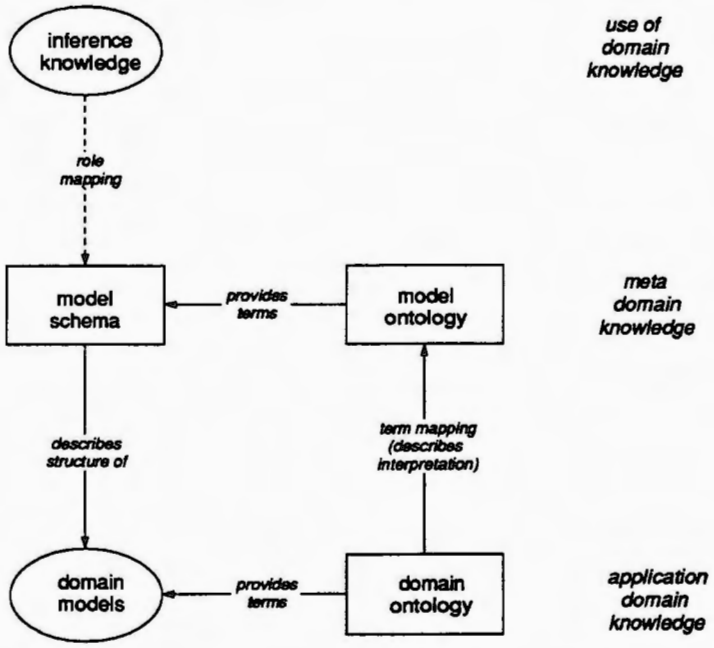


FIGURE 2: Relations between the various components of the domain knowledge in an expertise model

Fig. 2 summarizes the concepts that we have introduced in this section. A more powerful support for domain modelling is definitely one of the most important improvements over the original KADS-I framework.

3.2.2 Task knowledge The task knowledge specifies tasks. Knowledge about a task relates to the goal of the task, as well as to the activities that contribute to the achievement of that goal. Basically, as explained in Sec. 3.1, a goal of a task is a specification of the case model under construction. It is assumed, however, that task knowledge is in principle

independent of a domain and that references to the domain and the case model are through a task specific terminology only (see the description of roles in Sec. 3.3.2).

The notion of *task* is heavily overloaded in the AI and KBS literature. A common view [Puerta *et al.*, 1991] is to characterize a task by an I/O pair, and by some constraints on the relation between input and output posed by the task environment. In other fields of research concerned with the study of tasks (e.g., task analysis) a task is defined as a set of coherent activities that an agent performs in order to achieve a goal. In KADS-I a task was basically a goal-satisfaction procedure. This made it impossible to talk about a task independently of the method being used to achieve a goal. In CommonKADS both the goal and activity aspects of a task are maintained in the *task definition* and the *task body*, respectively. This is described in more detail in Sec. 4.3. Moreover, the relation between the two can be - but need not be - modelled as the application of a so called *problem solving method*, a notion which is introduced in Sec. 3.4.

3.2.3 Inference knowledge The inference knowledge specifies *basic inferences* that can be made in the domain knowledge. The role of inference knowledge is very similar to that of inference rules in (classical) logic. An inference rule in logic describes the ways in which axioms (domain knowledge) can be combined to derive new information. In what sequence or for what purpose such inferences are made is not described in the inference rule, but may be part of a mechanism embodied in a theorem prover. The difference between inference rules in logic and the CommonKADS basic inferences lies in the following features of CommonKADS inferences:

- Basic inferences operate on restricted parts of domain knowledge;
- Basic inferences are not necessarily truth preserving;
- Basic inferences implement an inference method that has a specific purpose in problem solving.

Inference knowledge is described in more detailed in Sec. 4.2.

The distinction between domain knowledge, task knowledge and inference knowledge is reminiscent of three views which are used in software engineering [Yourdon, 1989; p. 219]: the *data view*, the *time-dependent* or *control view*, and the *functional view*. Though not identical, the CommonKADS epistemological distinctions are similar.

3.3 Structuring a Knowledge Level Model

The role-limiting principle, which was introduced in Sec. 2, states that an agent can be modelled as imposing on its knowledge a structure, the parts of which play different specialized and restricted roles in the totality of problem solving. This structure is what in CommonKADS is the actual knowledge level model.[3] In the following subsections we describe first the structure within the three categories (Sec. 3.3.1), and then the structure between them (Sec. 3.3.2).

[3]Note that this is different from Newell's notion since it not only captures the content of the knowledge, but also the structure that emerges when the knowledge is being applied to a class of problems [van de Velde, 1991].

3.3.1 Intra-category Structure A CommonKADS expertise model consists of domain knowledge, task knowledge and inference knowledge. It is useful to describe its structure along the same lines.

- The structure on task knowledge is basically contained in the task body of the top-level tasks of an application. A task body contains a decomposition of a task into a set of sub-tasks which are themselves first-class tasks. The recursive decomposition of these subtasks leads to an overall *task structure* within the task knowledge. The task structure bottoms out in tasks which are not further decomposed (and are therefore called basic tasks). The basic tasks are linked to either basic inferences or to basic actions. The task body, in addition, specifies control over the tasks in the task decomposition.

- The structure on inference knowledge is the *inference structure*, well known from KADS-I [Breuker *et al.*, 1987]. The inference structure is the most primitive functional view on the application and describes the functional dependency among all basic inferences, abstracting from control. Note that the inference structure does not contain tasks or basic actions, only basic inferences.

- The primary structure on domain knowledge is provided by the domain models and their interrelationships through the inferences in which they play a role. Basic inferences derive relationships between case-specific information items that must ultimately relate to the case model. So the inference structure has a 'dual' view in the domain knowledge, called the *domain structure*, which contains the dependencies between the various domain models.

The task structure, inference structure and domain structure provide three useful but partial views on an application. Moreover, these views can be considered as relatively independent (the relative interaction hypothesis, Sec. 2). Therefore, they also provide for a useful decomposition of the modelling task.

3.3.2 Inter-category Structure Apart from structure within categories, a CommonKADS expertise model requires structure between the knowledge categories. We use the notion of *knowledge role* to describe the relations between knowledge elements of the various categories. Basically, a knowledge role is a reference to a knowledge item together with a role-specific vocabulary to talk about that knowledge item and its primary constituents.

Some concrete examples of knowledge roles are:

- A knowledge role for statements in the domain knowledge may point to a particular statement, for example `wire=broken`, and refer to it as a `hypothesis` and to its constituents `wire` and `broken` as `device` and `state`.

- A knowledge role for a relation among statements in the domain knowledge may point to a particular relation, for example `influences`, and refer to it as `causes` and to its constituents as `cause` and `effect`.

- A knowledge role for a task may point to a task `find-subpart` and refer to it - in a method specific way - as `generate`.

We distinguish two types of knowledge roles: *dynamic* knowledge roles and *static* knowledge roles. Dynamic knowledge roles point to knowledge elements that are being manipulated in the problem solving process: e.g., the input problem, the intermediate results, the solution. Static knowledge roles point to knowledge elements that are being used in the problem solving process, but that are not affected by it: e.g. causal knowledge about the normal functioning of a device, or a hierarchical decomposition of a device into its parts.

The above examples illustrate how knowledge roles allow for one category to refer to another one without making a commitment about the terms used in the latter. This is important to enable independence between and re-usability of parts of an expertise model.

3.4 Problem Solving Knowledge

3.4.1 Nature of Problem Solving Knowledge In CommonKADS the knowledge about how to model a particular application is called *problem solving knowledge*. Problem solving knowledge is knowledge about modelling in the sense of the modelling principle (Sec. 2). It is knowledge about how to organize and link the task, inference and domain knowledge related to an application in order to accomplish a given class of tasks within a given task environment.

Within the problem solving knowledge we distinguish two types: *problem solving methods* and *strategic knowledge*.

- A problem solving method captures a possible way of organizing the domain, task and inference knowledge. Its application therefore *generates* the knowledge structures within which the rational use of knowledge leads to meaningful problem solving behavior.
- Strategic knowledge specifies what kind of model of an application is adequate for a given task environment. The possible models are contained in the problem solving methods. Adequacy depends on availability of knowledge (epistemological adequacy) and the constraints on using or obtaining that knowledge (pragmatic adequacy).

Note that problem solving knowledge is not another knowledge category in addition to domain, task and inference knowledge. Problem solving knowledge is about a 'meta application', namely to come up with a good model in terms of application related knowledge. It would therefore be wrong and misleading to view it as a separate knowledge category. If this were done then there would be a category for knowledge about designing cars, writing programs, or curing patients. Clearly, we are using here a different categorization namely a teleological one. This categorization is orthogonal to the categorization of knowledge in task, inference and domain knowledge which is meant to be a purely epistemological one. As a consequence it is perfectly possible to analyze the problem solving knowledge in terms of the three basic categories of task, domain and inference knowledge. However, since the task here is always modelling and the method is always method selection and application it seems sufficient to provide only problem solving methods and strategic knowledge, basically the domain knowledge of strategic reasoning. The case model of strategic reasoning is formed by the application related knowledge.

Already in the KADS-I project it was recognized that the knowledge in the strategic layer is different from the knowledge in the other three layers. The strategic layer in

KADS-I -when it was present at all- was linked to all of the three other layers. This indicates that it is not really another layer 'on top of' the task layer, but that it should be orthogonal to all of the three other layers. At the KADS-I strategic layer there was no particular support for modelling. No attempt was made to provide generic components (the CommonKADS problem solving methods) or ways to understand the other layers from the general principles embodied in problem solving knowledge. Incorporating such facilities was a major aim of the CommonKADS integration effort.

3.4.2 Problem solving method Intuitively, a problem solving method describes a way (or ways) of solving a particular class of problems, i.e., a way of applying domain knowledge to perform a class of tasks. For example, the Cover-and-Differentiate method [Eshelman *et al.*, 1988; Schreiber *et al.*, 1992] describes how a particular type of diagnostic problem can be solved by exploiting causal knowledge. However, when one searches the literature for a precise definition of 'problem solving method', it becomes clear that there are different views.

A major aspect giving rise to different notions of problem solving method is the grain-size of the tasks that they apply to. In many approaches this task is a *general performance task* [Gruber, 1989], i.e. the overall task that a system is to perform. In earlier work by John McDermott and his group, where problem solving methods are coupled to complete execution shells, this is the only possible way [McDermott, 1988]. Also Clancey's Heuristic Classification method, although it can can be applied to tasks of various sizes, is usually applied to such larger tasks [Clancey, 1985b]. Chandrasekaran chooses a smaller grain size [Bylander & Chandrasekaran, 1988; Chandrasekaran, 1988]. Here problem solving methods are embedded in so called *generic tasks*. A problem solver for a performance task is decomposed into several such generic tasks, each applying their method (and representation). Communication between such generic task problem solving components is through message passing [Sticklen, 1989]. KADS-I takes a similar approach [Breuker *et al.*, 1987]: the general performance task (or *real-life task*) is decomposed into a number of generic tasks, such as prediction, classification, diagnosis, design, planning etc. The problem solving methods were embodied in the 'interpretation models' that describe a way of solving one such a generic task. In Components of Expertise [Steels, 1990] problem solving methods can be applied to any task. Yet smaller grain-size is chosen for the 'mechanisms'in PROTEGE-II [Puerta *et al.*, 1991] that are only the building blocks for complete problem solving methods.

Another difference is in the way methods are described. In general a problem solving method specifies a number of possible actions and some way of determining how these actions are to be ordered in time. In KADS-I methods are embodied in Interpretation Models, descriptions of inference structure and task-level control structure in which a number of problem solving methods are 'compiled out' to components in the four layer model. In the more recent KADS literature the notion of *knowledge differentiation* was introduced to indicate the decomposition of certain knowledge components, for example meta-classes and knowledge sources, into different subtypes. Knowledge differentiation appears to be the epistemic equivalent of problem solving methods. Newell [Newell, 1990] defines the concept of problem solving method in the context of the *problem space hypothesis*. At any instant an agent will pursue a goal in a problem space. The problem space contains states, a problem (i.e. an initial and a desired state), and a set of operators that

move from one state to another and control telling the agent how to navigate through the state space. Methods are viewed as goal structures, corresponding in a way to degenerate problem spaces: they are the result of applying operators in a state space while using the control information.

In CommonKADS we use the following notion of problem solving method:

> *A problem solving method is a prescription of the way in which a certain class of task definitions can be satisfied. The method may be* complete *with respect to a set of basic capabilities, i.e. prescribe all necessary actions in terms of the basic capabilities of the agent, or may be* incomplete *in the sense that the activities generated by the method my lead to new problems, which cannot be solved by basic capabilities and require the application of new problem solving methods.*

In our view a problem solving method specifies the relation between a task definition and a task body. A problem solving method is applied to a task definition and after a mapping of (generic) terms used in the method description onto the task specific terms, the body of the task can be instantiated from the method description. Whether all this is being done automatically within a system (as it was done in some experiments on Components of Expertise [Vanwelkenhuysen & Rademakers, 1990]) or manually by the knowledge engineer (i.e., knowledge differentiation) is irrelevant here.

In the next section, the framework is described in more technical depth, providing descriptions of each of the ingredients.

4 CommonKADS Conceptual Modelling Components

In this section we describe in more detail the different components of the CommonKADS expertise model. Fig. 3 shows the top-level decomposition of this model. A major distinction is made between parts of the model that concern application-related knowledge on the one hand, and problem solving knowledge on the other hand. The application knowledge encompasses the domain knowledge, task knowledge and inference knowledge. The problem solving knowledge consists of problem solving methods and strategic knowledge. Below we will discuss the various components of the expertise model and illustrate them using a very simple example of a diagnostic application in the domain of cars. The notation used in this section is a simplified version of the CommonKADS Conceptual Modelling Language (CML) [Schreiber, 1992].

4.1 Domain Knowledge

The domain knowledge (Fig. 4) basically contains all the knowledge about the system (*object system*) that an application is about. Application domain knowledge comes in two flavors: static -case independent- domain knowledge and dynamic knowledge related to the case (*case model*). The static domain consists of *domain models* which are coherent collections of *domain expressions*. They capture the general knowledge relevant to the domain of application. The dynamic knowledge is grouped in the case model. It contains

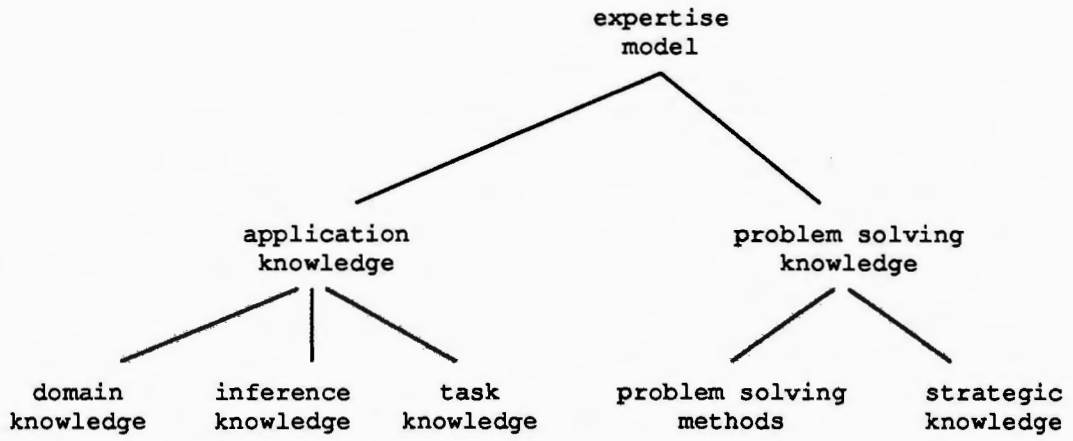


FIGURE 3: Major components of the expertise model.

a number of different types of information. First there are *case initial data* available before the reasoning even starts. For example, a description of the problem or a description of a particular configuration could be part of this initial case data set. The category *user input data* represents data about the case that the user may enter during problem solving. The category *case solution* represents the solution to the problem. Additional information that is case specific and that is derived during the reasoning process is also stored in the case model *intermediate case data*).

In addition to the case and domain models, the domain knowledge also defines the vocabulary from which these are built: the *domain ontology*. The domain ontology defines the terms that can be used in the domain and case models (*domain terms*), the knowledge types that these terms have (*domain typology*, e.g., concept, attribute, value, relation) and a isa-hierarchy of terms (*domain taxonomy*). The domain ontology closely corresponds to what was called in KADS-I the 'lexicon', 'concept-base' and 'concept hierarchy'.

Knowledge elements are represented in a modelling language that uses certain modelling terms and language constructs. For example in KADS-I these were derived from KL-ONE [Brachman & Schmolze, 1985] and consisted of *concepts*, *properties*, *relations* and *expressions*. This was too rigid and did not allow for multiple views on the knowledge. In CommonKADS modelling languages occur at two levels.

- First, there is the basic domain language, i.e., the one in which domain expressions are stated. This is captured in the *basic model ontology* and the *basic model schemata*. CommonKADS provides the KL-ONE ontology, but allows for extending it with other language constructs, if needed. For example, a language of equations could be useful. Table 1 shows the basic CommonKADS modelling terms and constructs.

- Second there are the modelling languages used to refer to statements within a single domain model. These are captured in the *domain model ontology* and the *domain model schemata* of the domain model. The interpretation of a domain model can be captured in additional *domain model axioms*, the assumptions that a model makes about the world. For example, particular theories of time or causation can be documented in this way and imposed on the domain knowledge.

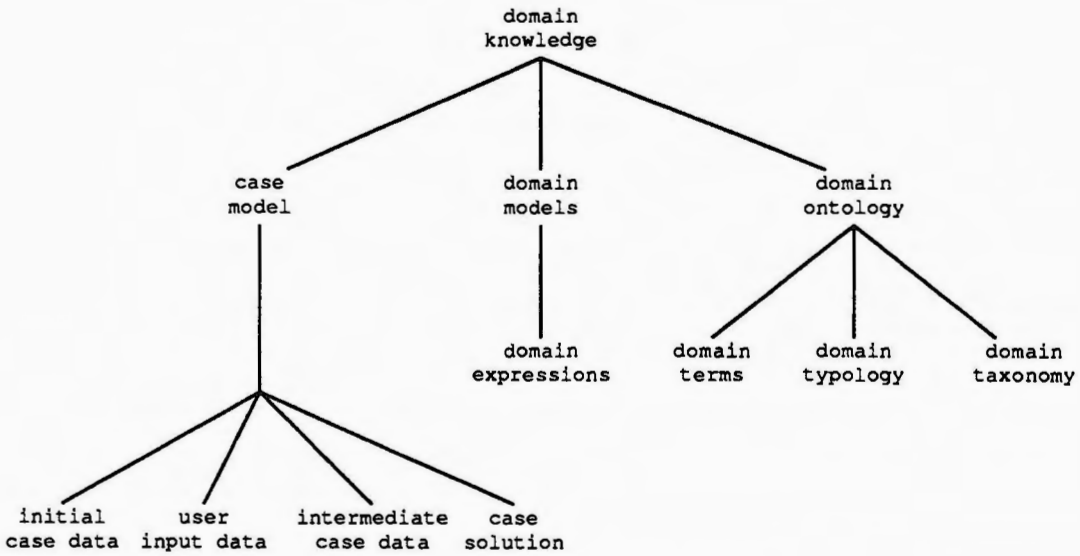


FIGURE 4: Components of the domain knowledge.

Fig. 5 shows the relation between these elements, which we collectively call *meta domain knowledge*, and the domain expressions. It shows how the notion of domain model allows to talk about collections of domain expressions in a coherent and well structured way, the domain models. These domain models capture required structures for the static knowledge roles of problem solving methods.[4]

Primitive term	Structure	Example
Concept	labels (atom)	The components of an audio system
Relation	relation between concepts or other entities	Sub-type hierarchy of components of the audio system
Attribute	the name of an attribute of a concept	color, size,
Value	the value that can be assigned to an attribute constant, concept or relation	red, big.
Expression	expression between features	power-switch=pressed
relation between expressions	expression relation expression	power-switch=pressed INDICATES power-state=on

TABLE 1: The CommonKADS basic modelling language

We can now define a number of example domain models and their meta description in terms of the basic domain language. Table 2 shows the domain ontology, i.e. the terms and typology of the domain.

Table 3 shows domain models for the example application and their structure in terms

[4]As such the meta domain knowledge is for the domain knowledge what the problem solving methods are for the task knowledge.

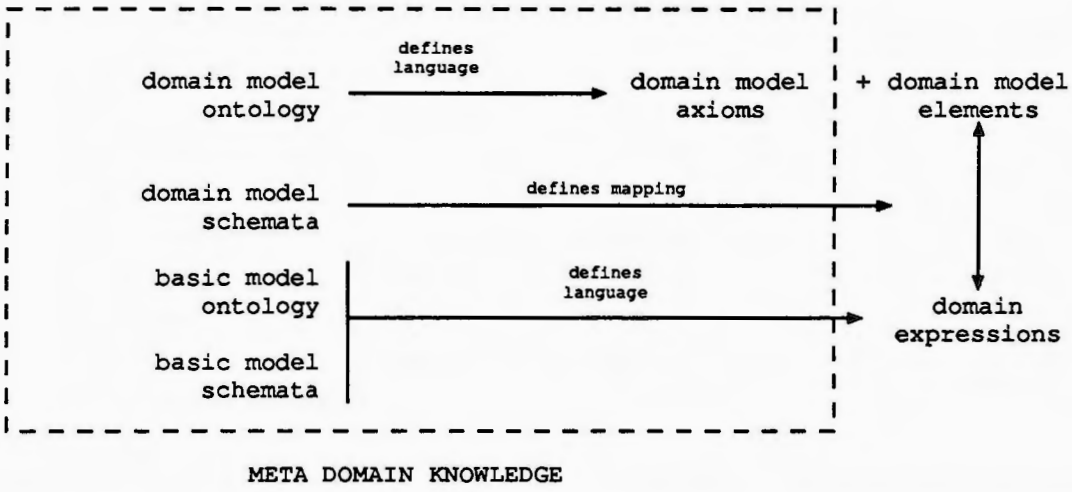


FIGURE 5: The meta domain knowledge related to the modelling languages for domain expressions and domain models.

Type	Domain Term
concept-label	no-gas battery-low no-gas-in-engine engine-does-not-run engine-stops
case-attribute	gas-dial battery-dial
value	zero medium full

TABLE 2: Domain ontology for simple car diagnosis

of the basic model terms. The names of the domain models suggest a certain type of knowledge that is represented in each model, but in actual fact these names are just pointers to a domain model. The representation chosen in the example is extremely simple. None of the domain entities has internal structure, states and parameters are represented by simple identifiers (concept-labels). A more elaborate representation would of course be possible.

In addition to the basic domain language CommonKADS provides the option to extend the modelling ontology with domain or task specific terms. For example, the term *explanation* is not part of the domain ontology, it is a term that *describes* a relation between domain concepts that has a certain significance in the context of diagnostic reasoning. Table 4 shows an example of an extended model ontology for the car-diagnosis domain.

It is important to note that the extended model ontology is different from the role descriptions (cf. meta-classes). The model ontology defines a vocabulary to describe the domain knowledge in abstract terms. It provides handles to the domain knowledge without having to specify all domain expressions of a certain type explicitly. It is essentially an extension of the data-modelling view with abstractions that allow easy linking to other

Domain Model	Domain Expression	Basic Model Schema
Fault classes	no-gas battery-low no-gas-in-engine	concept-label
Complaints	engine-does-not-run engine-stops	concept-label
Manifestations	gas-dial=zero battery-dial=zero	case-attribute = value
Explanation Model	no-gas-in-engine → engine-does-not-run no-gas-in-engine →engine-stops battery-low → engine-does-not-run	F:concept-label → C: concept-label
Behavioral Model	no-gas → gas-dial=zero battery-low → battery-dial=zero	F:concept-label → M:expression
Causal Model	no-gas → no-gas-in-engine	F1:concept-label →F2:concept-label

TABLE 3: Domain models with example domain expressions and the corresponding basic model schema for simple car diagnosis

Domain Model	Model Term	Domain Model Schema
Fault classes	fault-state(F)	fault-class: concept-label
Complaints	complaint(C)	complaint: concept-label
Manifestations	manifestation(M)	manifestation: case-attribute = value
Explanation Model	explain(F,C)	F:fault-class → C:complaint
Behavioral Model	has-manifestation(F,M)	F:fault-class → M:manifestation
Causal Model	causes(F1,F2)	F1:fault-class → F2:fault-class
Observation Model	observation(M)	M: manifestation M in case-user-input-data

TABLE 4: Extended modelling ontology and structure schema for simple car diagnosis

views.

4.2 Inference Knowledge

A reasoning process in a KBS consists ultimately of a number of inference steps that derive information about the system(s) that problem solving is about. These inferences make use of static domain knowledge and relate items of information in the case model.

In CommonKADS a primitive inference is basically a relation or a function between domain knowledge roles. An *inference relation* establishes a relation between the dynamic knowledge roles that it connects. For example the inference relation *cover* establishes a pair of complaint and fault-state in the role of symptom and cause, respectively, such that the fault-state explains the complaint. Fig. 6 shows a graphical representation of this basic inference. The dynamic roles (symptom and cause) are shown as white boxes, the static role causal model is shown as a grey box, while the box with rounded corners represents the relation that is established by the inference.

The definition of *cover* is given below.

inference-relation *cover(C,S)*
  goal
    ESTABLISH the cover relation between symptom and cause
  roles
    dynamic-roles
      S:symptom ↦ complaint

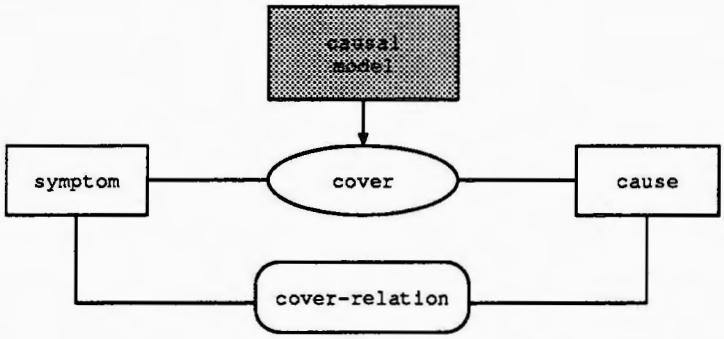


FIGURE 6: The inference relation *cover*.

C:cause ↦ fault-state
static-roles
causal-model: cause(C,S) ↦ explains(C,S) and cause(C,S)
inference-specification
(cause(C,S) ∨
∃ C1 cover(C,C1) ∧ cover(C1,S)) ⊢ cover(C,S)

Inference relations are non-directional, i.e. the direction of the inference is not specified. *Inference functions*, on the other hand, are directional: they specify which dynamic roles correspond to input and which ones to output roles. For example, the *compute* inference (as used in KADS-I) clearly has a directional nature (see Fig. 7).

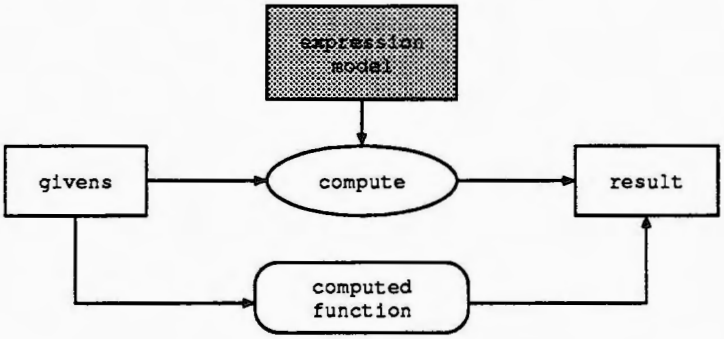


FIGURE 7: The inference function *compute*.

Below an example is given of an abductive inference function for the car diagnosis example.

inference-function *generate-by-abduction(S,H)*
goal
FIND a hypothesis H that explains symptom S
roles
input-roles
S: symptom ↦ complaint
output-roles
H: hypothesis ↦ fault-state

static-roles
implication-model: implies(H,S) $\mapsto$ explains(H,S)
assumptions
A1: implies(H,S) $\mapsto$ H $\vdash$ C
inference expression
implies(H,S) $\vdash$ hypothesis(H)

An *inference structure* is a structure in the inference knowledge and as such a model of a reasoning process (Fig. 8). An inference structure describes how individual inferences can be usefully related to each other. The inference structure thus specifies the inference capability or *competence* of the system.

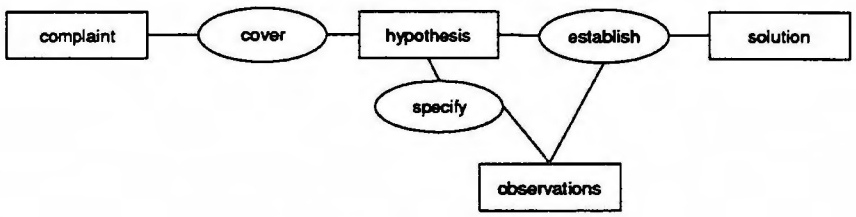


FIGURE 8: Inference Structure for the car example

4.3 Task Knowledge

The knowledge category of *task knowledge* describes how a goal can be achieved. A task is characterized by two parts: the *task definition* and the *task body*. The task definition describes *what* needs to be achieved. It is a declarative specification of the goal of the task. The task body describes *how* this goal can be achieved. It is a procedural program, prescribing the activities to accomplish the task.

The *task definition* consists of:

- *Goal:* A description of the goal that can be achieved through application of the task.
- *Case roles:* A definition of the dynamic knowledge roles that the task manipulates.
- *Task specification:* A description of the logical dependencies between the case roles of the task (e.g. what is true after execution of a task, invariants).

The *task body* has the following subparts:

- *Sub-goals:* The sub-goals that the task can generate.
- *Sub-tasks:* The sub-tasks that achieve the sub-goals.
- *Task expression:* The description of control over the sub-tasks to achieve the task.

We distinguish three different types of tasks:

- Composite tasks: tasks that are further decomposed in sub-tasks, e.g. diagnosis is decomposed into generate and test.

- Primitive tasks: tasks that are directly related to inferences. E.g. a primitive abstraction task could be the computation of all solutions of an *abstract* inference, given a particular data-set and a body of domain knowledge
- Transfer tasks: tasks of interaction with the world, i.e., the user. These tasks are not further specified in the expertise model but are part of the communication model [de Greef & Breuker, 1992].

Below two examples of task descriptions are given for the car diagnosis example. The first task represents the top-level diagnostic task. It is a composite task.

task *car-diagnosis(C,F)*

task-definition
goal:
FIND a fault-state F that explains complaint C ∧
all manifestations of F are observed
roles:
case-initial-input:
C: complaint
case-user-input-data:
O: set of observation
case-solution:
F: fault-state
task-specification:
explains(F,C) ∧
∀ M:manifestation has-manifestation(F,M) ⊢ observation(M)

task-body
sub-goals:
G1: FIND fault-state H with explains(F,C)
G2: TEST H such that
∀ M:manifestation has-manifestation(H,M) ∧
observation (M)
sub-tasks:
G1: generate(C,H)
G2: test(H)
control-roles:
hypothesis H: fault-state
task-expression:
REPEAT
generate(C,H)
UNTIL test(H)
RESULT(H)

The top-level task *car-diagnosis* invokes a sub-task *generate* to achieve one of its sub-goals. This task is defined below. Its goal is derived from the parent task. The control role hypothesis becomes an output of the task. The task body calls the basic inference *generate-by-abduction*. The task is therefore a basic task.

task *generate(C,H)*

task-definition
goal:

```
    FIND fault-state H with explains(H,C)
  roles:
    case-initial-input
      C: complaint
    case-solution
      H: fault-state
  task-specification:
    generate(C,H) ⊢ explains(H,C)

task-body
  task-expression:
    generate-by-abduction(C,H)
```

Fig. 9 shows the task structure implied by these two example task descriptions.

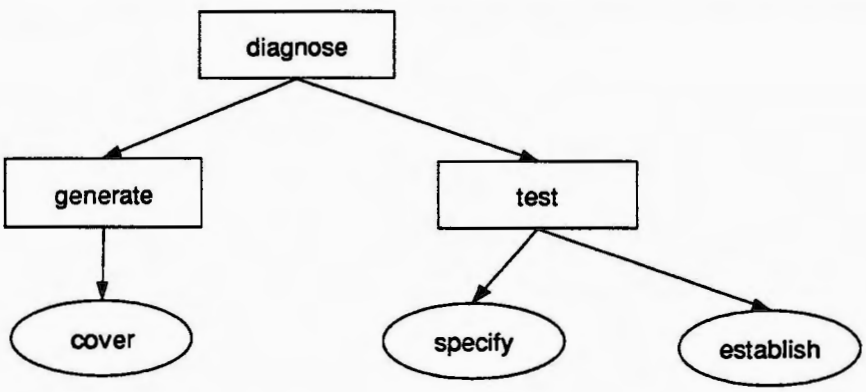


FIGURE 9: Task structure for the car example. Boxes represent composite tasks; ovals primitive tasks that map to inferences.

4.4 Problem Solving Knowledge

As explained in Sec. 3.4, problem solving knowledge consists of problem solving methods and strategic knowledge. In CommonKADS we view a problem solving method as the link between a task definition and a task body. Methods are applied to goals in the task definition and result in a decomposition of the task: the task body (Fig. 10).

A problem solving method is described as follows:

- *Goal:* a description of the goal that this method can in principle achieve.
- *Task characterization:* gives the abstract form that the the task specification of a task should have in order for the method to be in principle applicable.
- *Control roles:* any additional roles that the method introduces. These become part of the case intermediate data.
- *Sub-goals:* the sub-goals that the method sets up.
- *Definition:* a description of why the method works, what assumptions it makes and what it achieves.

- *Task expression schema:* a template for the task expression that can be used in the task definition of the task onto which the method is applied.

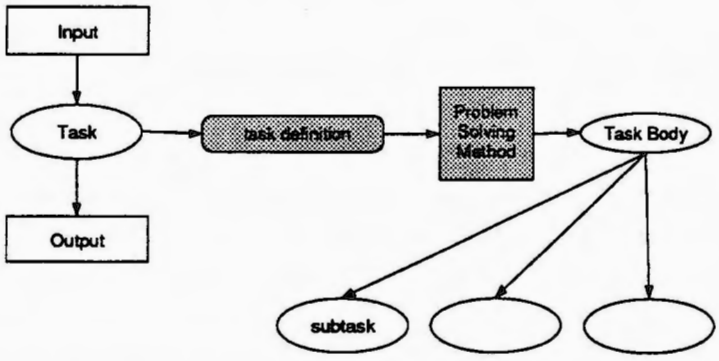


FIGURE 10: Application of a problem solving method to a task definition

Below we present as an example a quite general generate-and-test problem solving method. It is the method that could be applied in the car diagnosis example to arrive at the task bodies of the two tasks that were shown previously.

problem-solving-method *generate-and-test*
goal:
G: FIND(S:solution)
task-characterization:
criterium1(S) ∧ criterium2(S) ⊢ solution(S)
control-roles:
H: hypothesis → solution
sub-goals:
generate(problem, hypothesis)
test(hypothesis)
method-definition:
A1: ∀ x solution(x) ⊢ generate(x)
A2: ∀ x generate(x) ∧ test(x) ⊢ solution(x)
A3: ∀ x generate(x) ⊢ criterium1(x)
A4: ∀ x test(x) ⊢ criterium2(x)
A1 ∧ A2 ∧ A3 ∧ A4 ⊢ <P1> ∃ s solution(s)
task-expression-schema P1:
REPEAT
generate(C,H)
UNTIL test(H)
RESULT(H)

Basically the idea here is that the method can be applied to a task if the task specification can be interpreted as a conjunction of two criteria. The method then indicates two subgoals for generating and testing possible solutions. The method definition lists various assumptions that the method makes about the relation between these subgoals and the criteria in the task specification. Only if these assumptions are acceptable (either because they are true in the domain or it is justified to assume them) then the method should be applied. This means that the task body of the task it applies to can be instantiated

according to the task expression schema. After execution of this schema, then, a solution will be found.

The approach of making explicit what the method can achieve under which conditions was first explored in the competence theories of [Van de Velde, 1988]. Note that these assumptions can be translated into assumptions about the domain, depending on the way generation and testing are realized. For example, when generation is done from the subparts of the car, then it can be derived that the method is only competent to conclude about malfunctioning parts and, for instance not about faulty connections between them. Making these assumptions explicit the knowledge engineer gets an additional means of verifying the acceptability of his model.

The method definition as we have presented it allows one to check whether the method is in principle applicable. It does not say whether it is a good idea to do so. This depends on various factors related to the available knowledge and the task environment. For example, a generate and test method may be computationally too expensive even though possible, or a heuristic method that can propose erroneous solutions can be inadequate in a critical environment, and so forth. The characteristics of the task environment that determine adequacy of a problem solving method - and of a model in general - are called task features. The careful study of task features has only just begun and there does not seem to be conclusive theory about them. In CommonKADS we are approaching the problem in the context of the CommonKADS library, a library of generic models and re-usable model components. The task features are the indices into the library.

5 Conclusions

The aim of the research reported in this paper was to create a unified framework for knowledge modelling in knowledge-based system development. A careful study of the various knowledge modeling approaches revealed that different approaches are not incompatible, even though terminology is often very different. A major point of disagreement that remains is the interaction hypothesis in its strong form, that states that a knowledge level model cannot be an adequate specification of a computational system without the specification of the computational constructs. We have argued against the strong interaction hypothesis and introduced a much weaker form which still allows the distinction between an analysis and a design model that is so characteristic for KADS.

A second point where the modelling framework developed here differs from the KADS-I modelling approach is the dynamics of the problem solving process. Although the strategic layer in KADS-I allowed to model very flexible problem solving behavior in principle, in actual practice such flexible models are almost never built and no library support is available. The incorporation of the notion of *problem solving method* as developed at the VUB, widens the scope of the modelling methodology. Thus, models of expert problem solving built using the CommonKADS framework can be more dynamic than the largely static models in KADS-I.

We are further developing the CommonKADS framework in two directions. The first one is toward libraries of re-usable models, called the CommonKADS library, the second one concerns the use of formal models.

- The primary role of the CommonKADS library is to support a knowledge engineer

by providing generic models and re-usable building blocks. A secondary role of the CommonKADS library is to integrate the experience on modeling from the wider KBS community. Indeed, achieving a degree of re-usability is a common goal in knowledge engineering. Different views are around on how this can be realized, i.e., on what should be the nature of the re-usable components [Genesereth & Fikes, 1992; Gruber, 1992; Musen, 1991; Puerta *et al.*, 1991; Steels, 1992] and how the modeling process itself can be best characterized. As far as we know ours is the first comprehensive attempt to integrate these different views into a single library.

- In order to make our knowledge modeling framework more precise we are developing FML, a formal modeling language for CommonKADS expertise models. The advantages of a formal modeling language are in a common semantics, unambiguous models, ease of communication and the possibility to verify certain properties of the model or of the problem solving behaviors being modeled. This work extends previous results on the formalization of KADS models [van Harmelen & Balder, 1992] and their operationalization [ten Teije *et al.*, 1991].

Acknowledgments

Many colleagues at UvA, VUB and ECN contributed to discussions that led to this paper: Manfred Aben, Ameen Abu-Hanna, John Balder, Richard Benjamins, Bart Benus, Bert Bredeweg, Joost Breuker, Cuno Duursma, Frank van Harmelen, Wouter Jansweijer, Philip Rademakers, Johan Vanwelkenhuysen, Luc Steels, and Andre Valente. Rob Martil (Lloyd's Register), Olle Olsson and Klas Orsvarn (SICS) provided extensive comments on earlier versions.

References

AKKERMANS, J. M., VAN HARMELEN, F., SCHREIBER, A. T., & WIELINGA, B. J. (1992). A formalisation of knowledge-level models for knowledge acquistion. *International Journal of Intelligent Systems*. forthcoming.

BOOSE, J. H. (1985). A knowledge acquisition program for expert systems based on personal construct psychology. *Int. J. Man-Machine Studies*, 23:495–525.

BRACHMAN, R. J. & SCHMOLZE, J. G. (1985). An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9:171–216.

BREUKER, J. A. & WIELINGA, B. J. (1989). Model Driven Knowledge Acquisition. In Guida, P. & Tasso, G., editors, *Topics in the Design of Expert Systems*, pages 265–296, Amsterdam. North Holland.

BREUKER, J. A., WIELINGA, B. J., VAN SOMEREN, M., DE HOOG, R., SCHREIBER, A. T., DE GREEF, P., BREDEWEG, B., WIELEMAKER, J., BILLAULT, J. P., DAVOODI, M., & HAYWARD, S. A. (1987). Model Driven Knowledge Acquisition: Interpretation Models. ESPRIT Project P1098 Deliverable D1 (task A1), University of Amsterdam and STL Ltd.

BYLANDER, T. & CHANDRASEKARAN, B. (1988). Generic tasks in knowledge-based reasoning: The right level of abstraction for knowledge acquisition. In Gaines, B. & Boose, J., editors, *Knowledge Acquisition for Knowledge Based Systems*, volume 1, pages 65–77. Academic Press, London.

CHANDRASEKARAN, B. (1988). Generic tasks as building blocks for knowledge-based systems: The diagnosis and routine design examples. *The Knowledge Engineering Review*, 3(3):183–210.

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

CLANCEY, W. J. (1985a). Acquiring, representing and evaluating a competence model of diagnostic strategy. In Chi, Glaser, & Far, editors, *Contributions to the Nature of Expertise*.

CLANCEY, W. J. (1985b). Heuristic classification. *Artificial Intelligence*, 27:289–350.

DE GREEF, P. & BREUKER, J. A. (1992). Analysing system-user cooperation. *Knowledge Acquisition*, 4(1). Special issue 'The KADS approach to knowledge engineering'.

DE HOOG, R., MARTIL, R., WIELINGA, B., TAYLOR, R., & BRIGHT, C. (1991). Models in the kads-ii life cycle approach. ESPRIT Project P5248 KADS-II/WP1-2/TR/UvALR/018/2.0, University of Amsterdam, Lloyd's Register & Touche Ross Management Consultants.

DIEDERICH, J., RUHMANN, I., & MAY, M. (1987). KRITON: A knowledge acquisition tool for expert systems. *International Journal of Man-Machine Studies*, 26:29 – 40.

ESHELMAN, L. (1988). MOLE: A knowledge-acquisition tool for cover-and-differentiate systems. In Marcus, S., editor, *Automating Knowledge Acquisition for Expert Systems*, pages 37–80. Kluwer Academic Publishers, The Netherlands.

ESHELMAN, L., EHRET, D., MCDERMOTT, J., & TAN, M. (1988). MOLE: a tenacious knowledge acquisition tool. In Boose, J. H. & Gaines, B. R., editors, *Knowledge Based Systems, Volume 2: Knowledge Acquisition Tools for Expert Systems*, pages 95–108, London. Academic Press.

GENESERETH, M. & FIKES, R. E. (1992). Knowledge interchange format version 3.0 reference manual. Report Logic 92-1, Logic Group, Stanford University, CA.

GRUBER, T. (1990). The role of standard knowledge representation for sharing knowledge-based technology. Technical Report Report KSL 90-53, Knowledge Systems Laboratory, Stanford University, CA.

GRUBER, T. (1992). Ontolingua: A mechanism to support portable ontologies. version 3.0. Technical report, Knowledge Systems Laboratory, Stanford University, CA.

GRUBER, T. R. (1989). *The Acquisition of Strategic Knowledge*. Perspectives in Artificial Intelligence, Volume 4. Academic Press, San Diego.

LENAT, D. B. & GUHA, R. V. (1990). *Building large knowledge-based systems. Representation and inference in the Cyc project*. Addison-Wesley, Reading Massachusetts.

MCDERMOTT, J. (1988). Preliminary steps towards a taxonomy of problem-solving methods. In Marcus, S., editor, *Automating Knowledge Acquisition for Expert Systems*, pages 225–255. Kluwer Academic Publishers, The Netherlands.

MCDERMOTT, J. (1989). The world would be a better place if non-programmers could program. *Machine Learning*, 4(3/4):337–338.

MUSEN, M. (1991). Dimensions of knowledge sharing and reuse. Technical Report Report KSL-91-65, Knowledge Systems Laboratory, Stanford University, CA.

MUSEN, M. A. (1989). *Automated Generation of Model-Based Knowledge-Acquisition Tools*. Pitman, London. Research Notes in Artificial Intelligence.

NEWELL, A. (1982). The knowledge level. *Artificial Intelligence*, 18:87–127.

NEWELL, A. (1990). *Unified Theories of Cognition*. Harvard University Press, Cambridge, MA.

PUERTA, A., EGAR, J., TU, S. W., & MUSEN, M. A. (1991). A multiple-method knowledge-acquisition shell for the automatic generation of knowledge-acquisition tools. In *Proceedings of the 6th Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada*, pages 20.1–19. SRDG Publications, University of Calgary.

SCHREIBER, A. T., AKKERMANS, J. M., & WIELINGA, B. J. (1991). On problems with the knowledge level perspective. In Steels, L. & Smith, B., editors, *AISB–91: Artificial Intelligence and Simulation of behaviour*, pages 208–221, London. Springer Verlag. Also in: *Proceedings Banff-90 Knowledge Acquisition Workshop*, J. H. Boose and B. R. Gaines (editors), SRDG Publications, University of Calgary, pages 30-1 – 30-14.

SCHREIBER, A. T., WIELINGA, B. J., & AKKERMANS, J. M. (1992). Differentiating problem solving methods. In Wetter, T., Althoff, K.-D., Boose, J., Gaines, B., Linster, M., & Schmal-

hofer, F., editors, *Current Developments in Knowledge Acquisition - EKAW'92*, pages 95–111, Berlin/Heidelberg. Springer Verlag.

SCHREIBER, G. (1992). CML language constructs. ESPRIT Project P5248 KADS-II/T1.1/TR/UvA/023/1.0, University of Amsterdan.

STEELS, L. (1984). Second generation expert systems. *Future Generation Computer Systems*, 1(4).

STEELS, L. (1990). Components of expertise. *AI Magazine*. Also as: AI Memo 88-16, AI Lab, Free University of Brussels.

STEELS, L. (1992). Reusability and configuration of applications by non-programmers. Technical Report AI Memo 92-4, AI-Lab, Vrije Universiteit Brussel, Brussels.

STICKLEN, J. (1989). Problem solving architecture at the knowledge level. *Journal of Experimental and Theoretical Artificial Intelligence*.

TEN TEIJE, A., VAN HARMELEN, F., & REINDERS, M. (1991). Si(ML)^2: a prototype interpreter for a subset of (ML)^2. Technical Report ESPRIT Project P5248 KADS-II/T1.2/TR/UvA/005/1.0, University of Amsterdam & Netherlands Energy Research Foundation ECN.

VAN DE VELDE, W. (1988). Inference stucture as a basis for problem solving. In Kodratoff, Y., editor, *Proceedings of the 8th European Conference on Artificial Intelligence*, pages 202–207, London. Pitman.

VAN DE VELDE, W. (1991). Tractable rationality at the knowledge-level. In Steels, L. & Smith, B., editors, *Proceedings AISB'91: Artificial Intelligence and Simulation of Behaviour*, pages 196–207, London. Springer Verlag.

VAN HARMELEN, F. & BALDER, J. R. (1992). (ML)^2: a formal language for KADS models of expertise. *Knowledge Acquisition*, 4(1). Special issue: 'The KADS approach to knowledge engineering'.

VANWELKENHUYSEN, J. & RADEMAKERS, P. (1990). Mapping knowledge-level analysis onto a computational framework. In Aiello, L., editor, *Proceedings ECAI-90, Stockholm*, pages 681–686, London. Pitman.

WIELINGA, B. J. & BREUKER, J. A. (1986). Models of expertise. In *Proceedings ECAI-86*, pages 306–318.

WIELINGA, B. J., SCHREIBER, A. T., & BREUKER, J. A. (1992). KADS: A modelling approach to knowledge engineering. *Knowledge Acquisition*, 4(1). Special issue 'The KADS approach to knowledge engineering'.

YOURDON, E. (1989). *Managing the Structured Techniques*. Yourdon Press, Englewood Cliffs, New Jersey.