afcet
Association Française
pour la Cybernétique
Economique et Technique

arc
Association pour
la Recherche Cognitive

eccai
European
Coordinating Commitee
for Artificial Intelligence

EKAW 89

Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems.

Paris, July 89

Ed. J. Boose, B. Gaines, J.G. Ganascia

afcet
Association Française pour la Cybernétique Economique et Technique

arc
Association pour la Recherche Cognitive

eccai
European Coordinating Commitee for Artificial Intelligence

EKAW 89

Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems.

Paris, July 89

Ed. J. Boose, B. Gaines, J.G. Ganascia

CONTENTS

FOUNDATION

Knowledge Acquisition: some foundations, manual methods and future trends. 3
Shaw Mildred and Gaines Brian

The problem of knowledge extraction with human experts : A case of Expert Verbalizations. 19
Haers Monique and d'Ydewalle G.

Operation Presumption : Knowledge Acquisition by Induction. 34
Taki Hirokazu and Fujii Yuichi

The Science of Knowledge : A Research Program for Knowledge Engineering. 49
Addis Tom

The Validation of Knowledge Acquisition : Methodology and Techniques. 60
Benbasat Isak and Dhaliwal Jasbir

A philosophical basis for knowledge acquisition. 75
Compton P.and Jansen R.

Knowledge Acquisition : the continuum linking machine learning and expertise transfer. 90
Gaines Brian

A Framework for Knowledge Acquisition. 102
Klinker Georg

Acquisition, Classification and Formalisation of Software Specification Heuristics. 117
Niskier Celso and Maibaum Tom

Model Directed Inference Structure Acquisition. 128
Tong Xuejun and Konrad Erhard

REPRESENTATION

Integrating Task Demands and Problem-Solving Methods to Develop Useful Taxonomies for Knowledge Acquisition. 141
Woodward Brian

Situation Pattern Acquisition. Improves the Control of Complex Dynamic Systems. 160
Boy Guy and Caminel Thierry

Prospects for Conceptual Graphs in Acquisition Interfaces. 169
Eklund Peter and Kellett John

Knowledge Acquisition in the Domain of CNC Machining Centers : the MOLTKE Approach. 180
Althoff K. D., Kockskämper S., Traphöner R., Wernicke W. and Faupel B.

On 'Extracting knowledge from text': Modelling the architecture of language users. 196
Hirst Graeme and Regoczei Stephen

Knowledge Acquisition by Causal Model and Meta-Knowledge. 212
Charlet Jean and Gascuel Olivier

METAMETH. Methodology for Knowledge Engineering. 226
Nordbo Inge, Vestli Mette and Solvberg Ingebord

Towards Knowledge Acquisition in Natural Language Dialogue. 239
Schmidt Gabriele and Wetter Thomas

A Model for the Knowledge Analysis Process. 253
Twine Steven

Acquisition Of Task-Knowledge For An Adaptive User Interface. 269
Zinmeister Gaby and Hoppe Heinz Ulrich

LEARNING

EKAW-89 Tutorial Notes : Machine Learning. 287
Ganascia Jean-Gabriel

ISOLDE : A System for Learning Organic Chemistry through Induction. 297
Rose J. Royce and Gelernter Herbert

Towards Robust Expert Systems that Learn from Experience - An Architectural Framework. 311
Aamodt Agnar

A Knowledge-Based System that Supports Explanation-Based Learning Applications. 327
Chachere Lawrence

On Explanation and Bias in Inductive Concept-Learning. 338
De Raedt Luc and Bruynooghe Maurice

A Learning Expert System for Preventive Maintenance. 354
Dupas Rémy and Millot P.

A Learning System for INFERNO Inference Networks. 365
Gonzàlez José Carlos and Ramiro Francisco Alcaraz

Acquiring Knowledge about A Relation between Concepts. 380
Handa Ken-ichi and Ishizaki Shun

Malefiz - A Learning Apprentice System that Acquires Geometrical Knowledge about a Complex Design Task. 391
Herrmann Jürgen and Beckmann

Evaluating Interaction in Knowledge Acquisition : A Case Study. 406
Maulsby David, James Gregory and Witten Ian

A General Approach to the Use of Background Knowledge in a Numerical Induction Algorithm. 420
Mellis Werner

ARCHITECTURE

EKAW-89 Tutorial Notes : Interactive Knowledge Acquisition Tools. 437
Boose John H.

The Knowledge Dictionary : Storing Different Knowledge Representations. 448
Jansen Bob and Compton Paul

A Methodology for Analysing Modalities of System/User Cooperation for KBS. 462
De Greef Paul and Breuker Joost

ASKE : Towards Automatic Knowledge Acquisition. 474
Patel Jitu

A Graphical Framework for Knowledge Acquisition and Representation. 490
Kontio Jyrki and Lounamaa Pertti

From ETS to Aquinas : Six years of Knowledge Acquisition Tool Development. 502
Boose John, Bradshaw Jeffrey, Kitto Catherine and Shema David

A Mediating Representation to assist Knowledge Acquisition with MACAO. 516
Aussenac Nathalie, Frontin Jean, Rivière Marie-Hélène and Soubie Jean-Luc

Distributed Knowledge Acquisition as a Model for Requirements Elicitation. 530
Easterbrook Steve M.

Knowledge Elicitation for Image Interpretation.
Faure Claudie, Eskenazi Maxine, Frediani Simonetta and Saitta Lorenza 544

Methods for Knowledge Acquisition and Refinement in Second Generation Expert Systems. 554
Lavrac Nada and Mozetic Igor

Integrating Acquisition, Representation and Interpretation. 565
Linster Marc

SMAC : A Prototype of a Knowledge Modeling and Acquisition Tool for the Expert. 578
Moulin Sandrine

PROGRAM

Open Meeting – 3rd-4th July 1989

MRT 1, rue Descartes, 75005 Paris

Sunday 2nd July
5pm till 8pm Registration (MRT - 1, rue Descartes, 75005 Paris)

Monday 3rd July
8.30 till 9am Registration

9 am	Tutorial 1	Boose John:	EKAW-89 Tutorial Notes: Interactive Knowledge Acquisition Tools
10.30 am	Coffee break		
11 am	Tutorial 2	Shaw Mildred:	Knowledge Acquisition: some foundations, manual methods and future trends.
12.30 pm	Lunch		
14 pm	Tutorial	Wielinga Bob	
16 pm	Tutorial	Ganascia J.-G.:	EKAW-89 Tutorial Notes: Machine Learning
19 pm	Cocktail		

Tuesday 4th July

9 am	A1	Haers Monique:	The problem of knowledge extraction with human experts : A case of Expert Verbalizations
9.35 am	A2	Jansen Bob :	The Knowledge Dictionary: Storing Different Knowledge Representations
10.10 am	A3	Woodward B.:	Integrating Task Demands and Problem-Solving Methods to Develop Useful Taxonomies for Knowledge Acquisition
10.45 am	Coffee break		
11.15 am	B1	Boy Guy :	Situations Patterns Acquisition. Dynamic Systems
11. 50 am	B2	De Greef Paul:	A Methodology for Analysing Modalities of System/User Cooperation for KBS
12.25 pm	Lunch		
14 pm	C1	Eklund Peter :	Prospects for Conceptual Graphs in Acquisition Interfaces
14.35 pm	C2	Patel Jim :	ASKE: Towards Automatic Knowledge Acquisition
15.10 pm	C3	Taki Hirokazu:	Operation Presumption: Knowledge Acquisition by Induction
15.45 pm	Coffee break		
16.15 pm	D1	Kontio Jyrki :	A Graphical Framework for knowledge Acquisition and representation
16. 50 pm	D2	Rose J. Royce:	ISOLDE: A System for Learning Organic Chemistry through Induction
17.25 pm	D3	Althoff K. D.:	Knowledge Acquisition in the Domain of CNC Machining Centers: the MOLTKE Approach

EKAW-89

Foundations of Knowledge Acquisition Methodologies

KNOWLEDGE ACQUISITION: SOME FOUNDATIONS, MANUAL METHODS AND FUTURE TRENDS

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

INTRODUCTION

The problem of expertise elicitation from a skilled person is well-known in the literature of psychology (Nisbett & Wilson 1977, Broadbent, Fitzgerald and Broadbent 1986). Hawkins (1983) has analyzed the nature of expertise and emphasizes its severe limitations and dependence on critical assumptions which are often implicit. Dixon (1981) has surveyed studies showing that much human activity is not accessible to awareness. Collins (1985) has studied knowledge transfer processes among scientists and suggests that some knowledge may not be accessible through the expert, not only because he cannot express it, but also because he may not be aware of its significance to his activity. Bainbridge (1979, 1986) has reviewed the difficulties of verbal debriefing and notes that there is no necessary correlation between verbal reports and mental behavior, and that many psychologists feel strongly that verbal data are useless.

The main problems identified in accessing an expert's knowledge are:
- "Expertise" may be fortuitous. Results obtained may be dependent on features of the situation which the expert is not controlling.
- Expertise may not be available to awareness. An expert may not be able to transmit the expertise by critiquing the performance of others because he is not able to evaluate it.
- Expertise may not be expressible in language. An expert may not be able to transmit the expertise explicitly because he unable to express it.
- Expertise may not be understandable when expressed in language. An apprentice may not be able to understand the language in which the expertise is expressed.
- Expertise may not be applicable even when expressed in language. An apprentice may not be able to convert verbal comprehension of the basis of a skill into skilled performance.
- "Expertise" expressed may be irrelevant. Much of what is learnt, particularly under random reinforcement schedules, is superstitious behavior that neither contributes nor detracts from performance.
- Expertise expressed may incomplete. There will usually be implicit situational dependencies that make explicit expertise inadequate for performance.
- "Expertise" expressed may incorrect. Experts may make explicit statements which do not correspond to their actual behavior and lead to incorrect performance (Gaines 1987a).

WHAT IS KNOWLEDGE ENGINEERING?

The basic model for knowledge engineering has been that the knowledge engineer mediates between the expert and knowledge base, eliciting knowledge from the expert, encoding it for the knowledge base, and refining it in collaboration with the expert to achieve acceptable performance. Figure 1 shows this basic model with manual acquisition of knowledge from an expert followed by interactive application of the knowledge with multiple clients through an expert system shell.

This basic model has been greatly extended by the introduction of interactive knowledge acquisition and encoding tools allowing the expert to transfer knowledge directly to the system without an intermediary. Such tools can greatly reduce the need for the knowledge engineer to act as an intermediary, but, in most applications, they leave a substantial role for the knowledge engineer. As shown in Figure 2, knowledge engineers can:
- Advise the expert on the process of interactive knowledge elicitation
- Manage the interactive knowledge acquisition tools, setting them up appropriately
- Edit the unencoded knowledge base in collaboration with the expert
- Manage the knowledge encoding tools, setting them up appropriately
- Edit the encoded knowledge base in collaboration with the expert
- Validate the application of the knowledge base in collaboration with the expert
- Train the clients in the effective use of the knowledge base in collaboration with the expert by developing operational and training procedures

This use of interactive elicitation can be combined with manual elicitation and with the use of the interactive tools by the knowledge engineer rather than, or in addition to, the expert. The knowledge engineer can:
- Directly elicit knowledge from the expert
- Use the interaction elicitation tools to enter knowledge into the knowledge base

Figure 2 shows multiple knowledge engineers since the tasks above may require the effort of more than one person, and some specialization may be appropriate. Multiple experts are also shown since it is rare for one person to have all the knowledge required, and, even if this were so, comparative elicitation from multiple experts is itself a valuable knowledge elicitation technique.

Figure 2 shows the complexity of the knowledge engineer's role and the some of the support tools required. Figure 3 groups the support tools for editing, display, encoding and validation of the knowledge bases into an *Expert System Support Environment* (ESSE), and lumps the various forms of knowledge base together. It shows the overall structure of a knowledge-based system as a central knowledge base interacting through the knowledge acquisition tools, the expert system support environment, and expert system shell, with a user community of experts, knowledge engineers and clients.

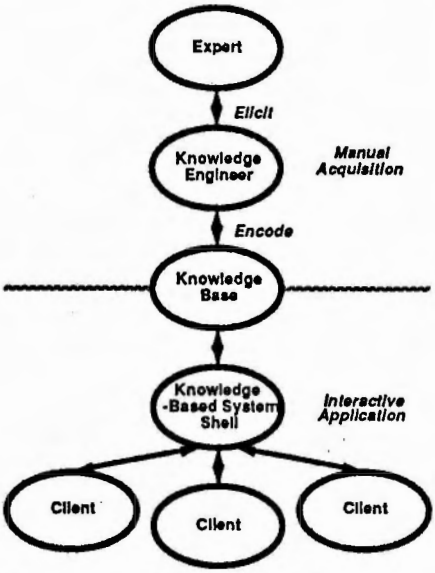


Fig.1 Basic model of manual knowledge acquisition

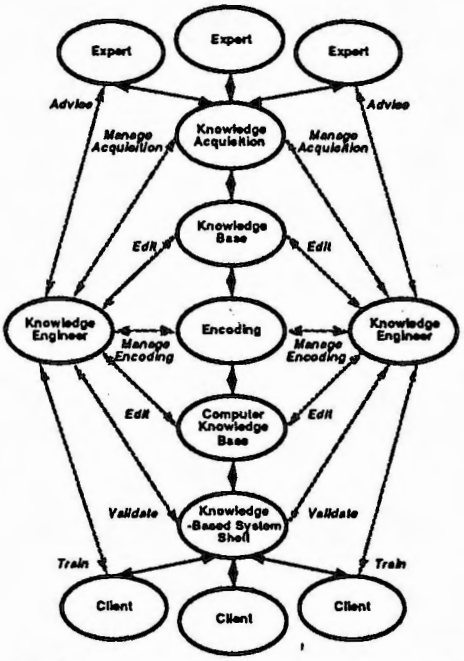


Fig.2 The knowledge engineers' roles in interactive knowledge acquisition

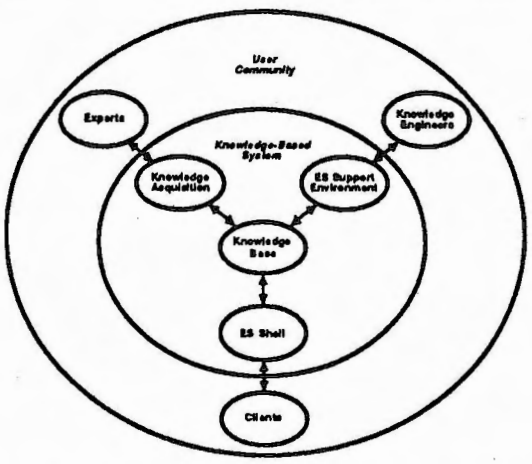


Fig.3 Major components of a knowledge based system

The notion of a user community is important in drawing attention to another trend in knowledge-based systems, that the distinction between 'experts', 'knowledge engineers' and 'clients' is often one of variable roles rather than hard and fast job definitions. The expert may be a major client for the system, or may play a major part in knowledge engineering, or both. Some clients may customize or update the system in the role of knowledge engineers or extend its capabilities in the role of experts.

MANUAL METHODS OF KNOWLEDGE ACQUISITION

The process of manual knowledge acquisition may be divided into phases: conceptualization of the domain, acquisition of knowledge and the analysis of results. Further phases in the development of the system include the implementation on a computer and the debugging of the operational system. . Figure 4 shows these phases of knowledge acquisition.

Domain organization involves the identification of functional goals and expectations, and the identification of sub-goals and tasks. In the preliminary knowledge acquisition stage it is necessary to identify knowledge acquisition needs and resources for each, decide how many experts are needed, decide on the number of sessions, how much time is required from each expert, how many knowledge engineers are required, what

types of knowledge acquisition will be used, the preliminary knowledge base design, and an estimate of the size and scope.

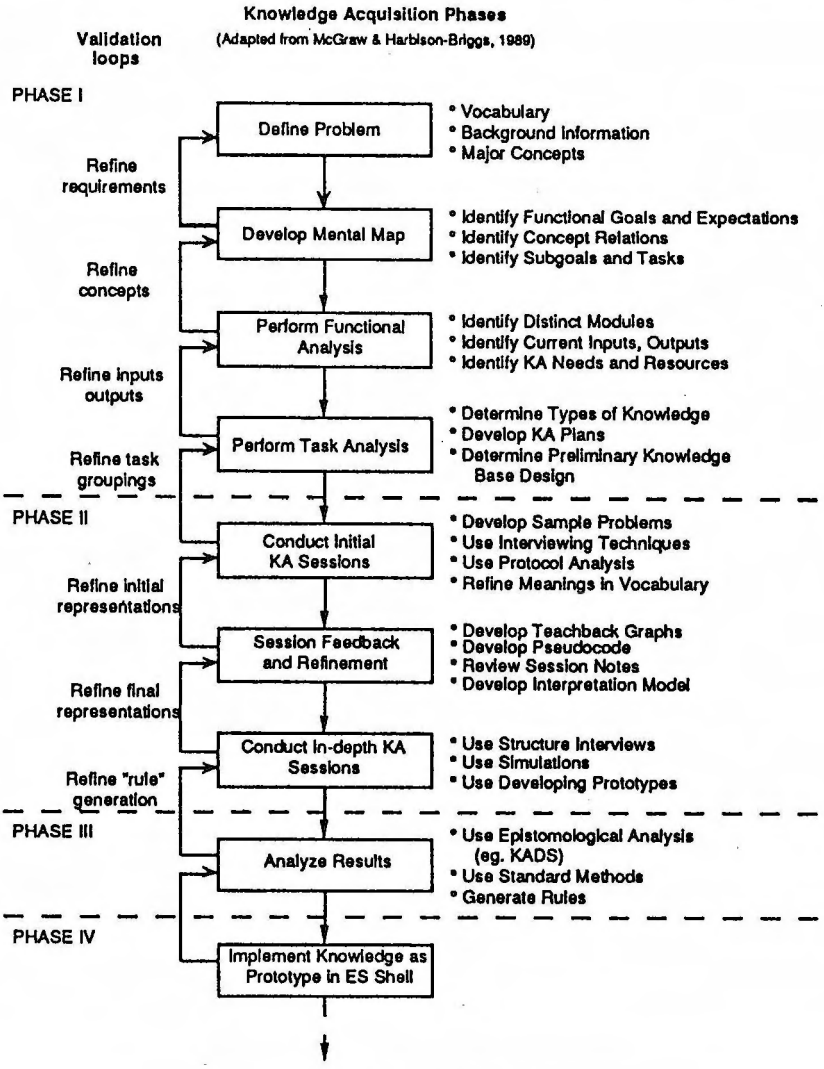


Fig.4 Phases in Knowledge Acquisition

The acquisition of knowledge needs the expert and the knowledge engineer to identify the general heuristics, the procedures, tasks, concepts and vocabulary. It is also necessary to identify the decision-making heuristics, and the analogies and problem-solving techniques used in the domain.

At the pre-implementation phases there are various techniques for domain organization such as generating concepts by brainstorming (van Gundy, 1981), the Crawford Slip method (Rusk and Krone, 1984) or Q-methodology (McKeown & Thomas, 1988), and organizing concepts into groups with the expert identifying common threads and providing labels while the knowledge engineer makes records in the form of lists, notes, diagrams, charts, a knowledge dictionary or glossary using audiotape, videotape, whiteboard or a wordprocessor.

Two examples of tools for domain organization are Cognosys (Woodward, 1988) and Knowledge Acquisition Design System (Breuker & Wielinga, 1987). In Cognosys the expert provides a description statement of overall domain and a list of general objectives which must be met in order to solve problems or make decisions successfully with the domain. He must the produce short paragraphs which are descriptive of the tasks required to meet each objective. These may be articles provided by the expert, such as case studies, text books or operating manuals. The knowledge engineer takes these and analyses the tasks into categories. He also completes the concept description categories and clarifies and elaborates such things as temporal relationships, event and state sequences, goal specification, cause-effect relationships, inferences, hierarchical relationships. It is important at this stage to clarify the domain boundaries with the expert and discuss specific problems in the

domain. He may identify other sources of domain knowledge, before he identifies knowledge acquisition methodologies and tools, report formats and inference processes.

The KADS methodology deals with the whole process from domain organization to developing a complete system. It breaks the domain organization into stages where the problem is decomposed into component parts and the analysis of knowledge is separated from its machine implementation. Data is then collected and tested to modify the framework. The three refinement cycles are applied: orientation which includes the acquisition of vocabulary, the assessment of characteristics, and the selection of the model and a test for problem feasibility; problem identification in which structures of domain concepts are uncovered, the functional analysis of the prospective expert system and task analysis; and problem analysis where the analysis of the user and operational environment results in the basic architecture of the system. At this point knowledge primitives are identified such as objects, knowledge sources, structures, strategies and models. Interpretation models are produced using a top-down model-driven approach which may include notions of causality, time, space, change and task-dependent concepts.

Building interpretation models is done by knowledge identification and knowledge conceptualization where knowledge primitives are ordered using relationships such as is-a, part-of, causes, depends-on, consists-of, defines, and several types of analysis are performed. The system is then coded using guidelines laid down.

SOME TECHNIQUES FOR THE MANUAL ACQUISITION OF KNOWLEDGE

Interviewing (McGraw & Harbison-Briggs, 1989) is used to identify tasks and major concepts, to structure and refine already-acquired information. Unstructured interviews are appropriate when the knowledge engineer wants to explore an issue or the goal is to establish rapport and to get a broad view. It facilitates the description of the domain in a way that is easy for the expert to understand. However, the data acquired is often unrelated and difficult to integrate due to its lack of structure. It may not allow gathering of specific knowledge and takes time and training to do well. Also, similar questions may be asked in future sessions and this may annoy the expert. Alternatively, structured interviews have the advantages of forcing an organization on the interview and being very goal-directed. They attempt to remove distortion from experts subjectively and allow better integration of material after the interview. They force the expert to be systematic, and the knowledge engineer is able to identify gaps in the knowledge which act as a basis for further questions. The purpose of the session is clear to the expert. However, they need more preparation by the knowledge engineer who must study the background material extensively.

During the interview, of any kind, the knowledge engineer must decide on a note taking approach, for example, video- or audio-taping, each of which has advantages and disadvantages. Both open and closed questions can be used. Open questions are broad and place few constraints on the responder. They are used for determining scope of understanding, response certainty, and the models used. These questions like *what, why, how* allow the expert to express information the knowledge engineer does not know about. He can obtain the expert's vocabulary, concepts, frames of reference, and can help with explanations and underlying theory. Closed questions such as *who, where, when, which* tend to set limits on the type, level and amount of information the expert provides since they often provide a choice of alternatives such as a bipolar or multiple choice response. These are used more for clarifying or probing questions or as feedback as they are less time consuming for specific information. Note-taking is much easier but sometimes one can get too little information as they may stop the expert from volunteering information. They requires an excellent command of vocabulary and concepts from the knowledge engineer. A balance of open and closed questions is optimal.

Two examples of semi-structured interviewing techniques are the Knowledge Acquisition Grid (LaFrance, 1987) and the Teachback technique of Johnson and Johnson (1987). In the Knowledge Acquisition Grid a matrix of knowledge types and forms is used:

Question Types	Forms of Knowledge: layouts	stories	scripts	metaphors	rules-of-thumb
grand tour					
cataloging categories					
ascertaining attributes					
determining inter-connections					
seeking advice					
cross-checking					

Examples of a *grand tour* are distinguishing boundaries of the domain, perspective, goals organization, e.g. Please describe the kinds of things you do; of *cataloging* may be an organized taxonomy of expert terms and concepts, e.g. How would you group and subgroup these things?; of *ascertaining attributes* are distinguishing features and range of values, e.g. Describe similarities and differences of concepts; of *determining interconnections* are uncovering relations and causal model(s), e.g. Why?; of *seeking advice* include revealing strategies, e.g. Which conditions warrant which actions?; and of *cross-checking* are validation concerns, e.g. knowledge engineer plays naive, devil's advocate, poses hypothetical situations and asks How sure are you? Are there exceptions?

The Teachback technique is based on Pask's (1975) *conversation theory* and on Systemic Grammar Networks from Linguistics. This method attempts to formalize a functional approach to grammar. It is an intermediate representation used to teach back to the expert the knowledge acquired in the interview. It is a *mediating* representation which is a machine language independent formalism between verbal data and software development environments. It is used as a tool to help proceed on early elicitation and conceptualization for synthesizing knowledge from talk with experts. It is mainly used as a means to prepare explicit descriptions of knowledge for the use of the development team to construct an associative network, graphs composed of nodes joined by links. It is said to be a generative form which captures all relevant information but excludes absurdities.

Two examples of Protocol Analysis, which are widely used, are Verbal Reports as Data (Ericsson & Simon, 1984) and the Specification of Expertise (Johnson, Zaulkernan & Garber, 1987). Ericsson's method is a well-known technique and one of the few established ways to elicit knowledge. It uses concurrent and retrospective verbalization for elicitation. *Concurrent report* reveals the sequence of heeded (attended to) information and *retrospective report* reveals the sequence of specific cognitive processes. A think-aloud transcript is then encoded into segments (assertions propositions, reading, doing arithmetic, visual searching, long-term memory recall, generalizing combinations) and relations of code are derived from verbs, propositions and adjectives. "Arguments" are often nouns and nouns phrases and the task analysis defines *a priori* a space of possible encodings representing information relevant to the task.

The method is based on the facts that:

- heeded (attended to) information is identified by analysis of corresponding verbalization through inference and task analysis
- the original encoding should reflect as closely as possible the verbalized information, preferably by automated or semi-automated means
- encoding uses a problem space approach (behavior is a search through a problem space involving the step-by-step accumulation of knowledge)
- the model distinguishes information directly stored in memory (facts, experiences, perceived events and behavior in past situations) from information generated or produced by inference (reactions and behavior in hypothetical situations)
- the criteria for inferring underlying cognitive processes from thinking aloud protocols ar relevance to task analysis and consistency with preceding verbalizations
- information in focal attention is vocalized directly or is encoded, recovery implies decoding
- segments are analyzed using syntax, pauses and intonation
- each segment is independent (the context is ignored for the reason cited above)
- encoding involves finding a category expressing the same information as the segments

The advantages of this method are that it can provide a complete record using both concurrent and retrospective reports then after definition of problem space and segmentation of the protocol the analysis can be automated or the task made much easier for the coder as it focuses the coder's attention on the segments. It requires consistency and explicitness of underlying assumptions, vocabulary and inference rules and the automated encoding lessens the risk of incorrect inference. A comparison of talking aloud protocols shows a very high consistency for practiced skills and successive protocols show stable task strategies and cognitive representations. It is an appropriate method for reporting specific, recent, episodic memories. On the other hand, retrospective reports may leave out important detail and include inferences and generated information (plausible explanations). It is often difficult to tell if the expert pays attention to the knowledge engineers's instructions. The expert cannot report cues allowing recognition of stimuli since these are in long term memory and his short term memory retains goals and intentions longer than heeded information. There are also incompleteness problems. Thought in non-oral form is much faster, for example, the knowledge engineer cannot instruct a juggler to slow down!

The Specification of Expertise (Johnson, Zaulkernan & Garber, 1987) is one way of using the material gained from protocol analysis. It is a psychology based manual technique in which the expert thinks aloud while working. It encompasses a syntactic and semantic analysis of the protocol using operations, episodes, and data cues. Expertise is characterized by semantic categories, actions abilities, goals, conditions, strategies and solutions. It is based on the facts that thinking is not directly observable, especially that of expert with considerable compiled knowledge and practice and that expertise is characterized by generativity and is a kind of knowledge that is used to perform a task. Part of this knowledge is represented by a set of abilities for problem solving. The representation or specification of expertise consists of: possible solutions and components of solutions; relevant information or data needed; goals for solutions; permissible ways to move between intermediate solutions and the abilities needed to make a transition.

ANALYSIS

Many existing statistical and clustering methods can be applied to the analysis of structured material such as factor analysis, Principal Component analysis or multidimensional scaling techniques. In recent years some

machine learning techniques have been found to be successful such as induction. There is no neat knowledge acquisition method currently for analyzing unstructured interview data. Data is normally long and complex and arranged in a non-standard format, and moreover, expert testimony usually relies on unspecified assumptions, background meanings and tacit knowledge. Pidgeon, Turner & Blockley (1989) have identified some issues that arise:

- Data is qualitative not quantitative
- Behavior and expertise are contextual
- Subjective orientation of the subject matter
- Complexity of expert knowledge
- Critical role of tacit knowledge
- Discovery of theory by testing hypotheses
- Data driven approach to elicitation
- Analysis is an interpretation of the expert's knowledge
- The knowledge engineer is a knowledge acquisition tool
- Elicitation and representation "fit" the phenomenon
- Results reflect and represent the intended meaning

The advice they give is to build as soon as possible after the interview and aim to arrive at a form of labelling which accurately characterizes the concepts, decision rules and strategies of the expert. The steps are:

1. Begin with first paragraph and ask what categories, concepts, or labels are needed in order to account for the phenomena of importance in the paragraph;
2. Record the label on a file card (or hypercard, or outliner) with a short description of the feature, and reference to the paragraph or line number;
3. Repeat for all paragraphs;
4. Recognize and list recurrences of labels;
5. Refine labels to produce appropriate concepts from all occurrences, then complete a written definition of the concept;
6. Definitions should contain descriptions of data patterns, generalizations, new categories, and links between categories;
7. Group sets of related concepts.

This process results in a set of fully saturated core primitives together with definitions, propositions linking categories, and graphical representations of relations among primitives.

FUTURE DEVELOPMENTS AND TRENDS IN KNOWLEDGE ACQUISITION

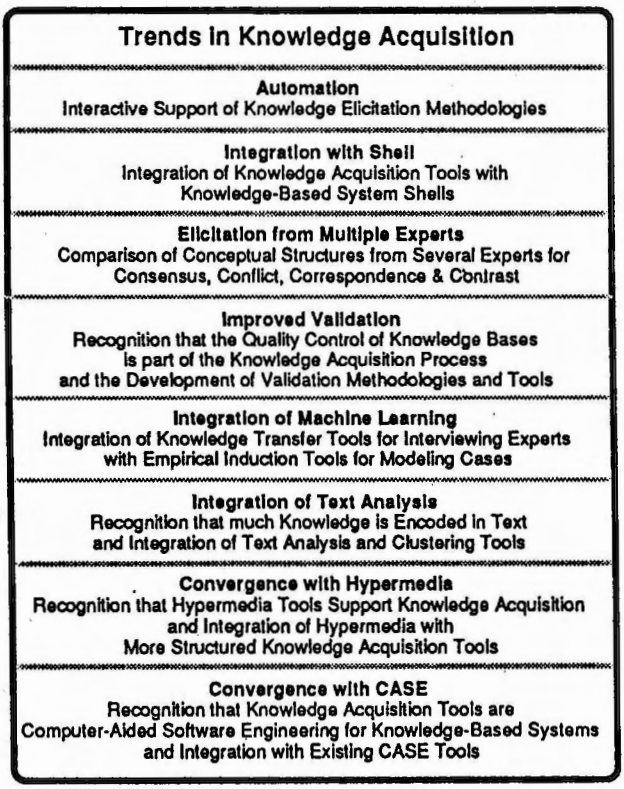


Fig.5 Significant Trends in Knowledge Acquisition

The initial development of knowledge acquisition techniques in the 1970s was based on human interviewers debriefing experts about their skills. The development of computer-based tools supporting knowledge acquisition in the 1980s has been based on the computer-aided application of specific techniques such as construct and conceptual mapping, behavior modeling and text analysis. However, neither manual nor tool-based approaches have yet made a significant impact on the knowledge acquisition bottleneck which continues to impede the development and effective application of knowledge-based systems (Gaines 1987a,b). Current research is attempting to establish a new paradigm for knowledge acquisition, one that recognizes the wide diversity of forms and sources of knowledge, its essential dynamism making it subject to contradiction and change, and the constructive nature of knowledge acquisition whereby it is often the knowledge acquisition process itself that creates knowledge. Figure 5 reflects some of the current trends in knowledge acquisition.

Steels (1986) has suggested that we are entering an era of *second generation* expert systems, "able to combine heuristic reasoning based on rules, with deep reasoning based on a model of the problem domain. This solves a number of major problems of current expert systems, most importantly the problem of knowledge acquisition: second generation expert systems can learn new rules by examining the results of deep reasoning." If one generalizes this concept and thinks of second generation systems as having *embedded knowledge acquisition* using a variety of sources including those suggested by Steels, then it captures the thrust of current research: to integrate knowledge acquisition and performance systems. The zeroth generation of manual knowledge acquisition techniques led to the first generation of computer-aided tools, and this is now leading to a second generation of integrated knowledge support systems.

Gaines (1988a) has suggested that second generation knowledge acquisition targeted on applications in the 1990s have to:

- Integrate a wide variety of weak methods encompassing the diversity of forms and sources of knowledge
- Integrate conflicting sources of knowledge to cope with essential contradictions that cannot be replaced by an artificial and misleading 'consensus'
- Integrate knowledge acquisition with system operation to facilitate a heavy activity of continuing system maintenance and upgrade as knowledge changes
- Integrate knowledge representation in the acquisition and performance systems, retaining the flexibility for new representations to be used that more accurately carry the knowledge

Thus, second generation developments have two objectives that may come into conflict:

- To integrate closely a variety of techniques and technologies to cope with the diversity of knowledge
- To provide flexibility within the integration framework for new techniques and technologies to be incorporated to cope with the dynamics of knowledge

AUTOMATION: KNOWLEDGE ACQUISITION AS INFORMATION TECHNOLOGY

To examine the technological support for second generation integrated knowledge acquisition systems it is necessary to have some model of the wide range of technologies underlying computing. The computer itself is only one component in an elaborate and highly differentiated infrastructure which has grown through a succession of generations of computers, each of which represents a major change in computing technology (Gaines & Shaw 1986). During the eight year span of each computing generation, revolutionary changes have taken place that correspond in magnitude to those taking place over some hundred years or more in other industries. The first generation commenced in 1948 and we are now in the sixth generation spanning 1988 through 1995.

The generation structure of information technology can itself be analyzed in terms of the knowledge flows involved. Like other basic technologies, the growth of computing has an infrastructure comprised of the envelope of learning curves in successive underlying technologies. Figure 6 shows this structure through the zeroth through fifth generations projected to the sixth and seventh. Each technology has a learning curve in which a *breakthrough* leads to a phase of: *replication* in which the results are duplicated; *empiricism* in which rules for design are derived from experience; *theory* in which generative principles are derived; *automation* in which the theory is made operational; and *maturity* when the knowledge becomes proven, widely available and widely used. The qualitative changes that have given computing its distinct character in each generation appear through the tiered succession of learning curves of higher level technologies based on the lower level developments:

- the initial breakthrough in *electronic device technology* leading to the zeroth generation is placed at 1940 about the time of the Atanasoff and Berry experiments with tube-based digital calculations
- the first generation breakthrough was the introduction by Mauchly and Von Neumann of stored program and subroutine concepts around 1947 which led to the transition from the ENIAC to the EDVAC designs—this detached computing as a separate discipline from electronics by substituting software for hardware in a *virtual machine architecture*
- the second generation breakthrough was to bridge the gap between machine and task through the development of *problem-oriented languages* such as FORTRAN around 1956
- the third generation breakthrough was to bridge the gap between the computer and the person with the development of interactive time-shared computers in 1964 allowing close *human-computer interaction*
- the fourth generation breakthrough was in the early 1970s with developments in expert systems based on knowledge-processing—such *knowledge-based systems* have the capability to store information through its inter-relations and make inferences about its consequence
- the fifth generation breakthrough was in 1980 with developments in machine learning and *inductive inference systems*
- the growth of robotics will provide the next breakthroughs in which goal-directed, mobile computational systems will act autonomously to achieve their objectives—the breakthrough into the sixth generation era commencing in 1988 is one of *autonomous activity systems*

• the interaction between these systems will become increasingly important in enabling them to cooperate to achieve goals—the seventh generation era commencing in 1996 will be one of *socially organized systems.*

The generations of knowledge acquisition systems are shown at the top of Figure 6—the current sixth generation of information technology is also marked. It coincides with the second generation of knowledge acquisition systems and this four generation gap indicates an analogy between the state of knowledge-based systems development now and that of data processing 32 years ago. We do indeed have a similar situation to the 1950s in computing when software development was new with problems of shortage of skills, unrealistic schedules and budget over-runs.

The detailed infrastructure for the development of the second generation of knowledge acquisition systems may also be seen in Figure 6. At the three lowest levels, the hardware machines architectures and languages are all mature technologies posing no problems. Human-computer interaction has entered its phase of automation with the increasing availability of machine-independent interactive systems such as X-windows. Knowledge-based systems have entered the theoretical development phase with increasing depth and structure to knowledge representation and inferencing schema. Inductive inference systems have entered the empirical phase where there are beginning to be available a range of usable tools. Autonomous activity systems are still in the research phase and the lack of experience and foundations will restrict the development of knowledge acquisition systems for goal-directed, planning activities during this generation.

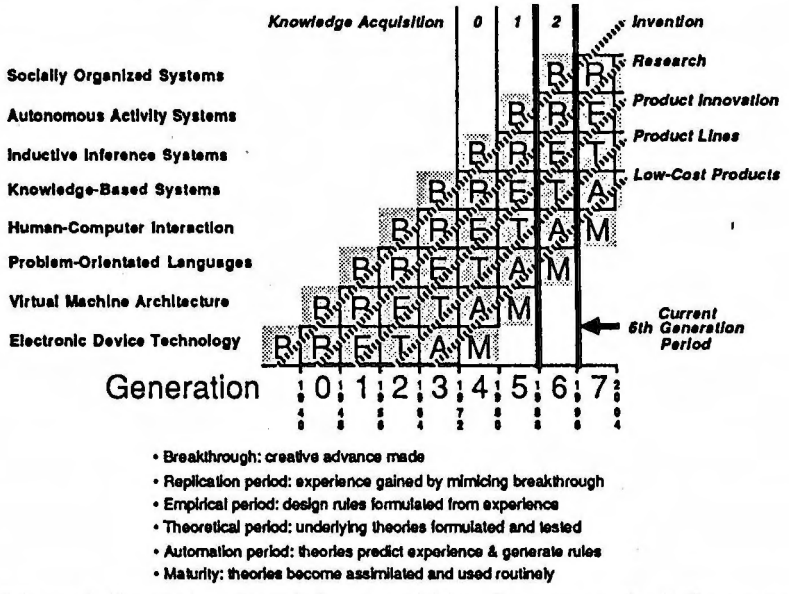


Fig.6 Second Generation Knowledge Acquisition Systems in the Infrastructure of Information Technology

Thus overall, we should expect the emphasis in second generation knowledge acquisition systems to be on the use of high-speed graphic workstations, programmed in high-level languages, using high-quality user interfaces to provide effective knowledge representation, some inductive inference and little or no capability to deal with planning.

INTEGRATION: KNOWLEDGE ACQUISITION AS PART OF A SOCIAL PROCESS

Hawkins (1983) has abstracted from industrial experience in developing mineral exploration expert systems and proposed a model of human expertise relevant to expert systems. Figure 7 presents a summary of the essential features of this model:

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

Thus, in Hawkins' model, the client plays an active role in further developing the model by providing more data until he or she is satisfied with the model and consequent advice. Expert advice giving and taking is part of a cycle of negotiation around a process of model formation. The particular importance of this model of expert performance is the emphasis it places on the embedded knowledge acquisition process. The expert is managing a data collection, modeling and decision cycle within a consultation. As Figure 7 shows, this implies that the expert is able to enhance his or her collection of case histories, models and advisory strategies through the consultation. Skilled performance is not the static utilization of expert knowledge and skills in Hawkins' model—it is instead a microcosm, and an essential part of, the wider processes of knowledge acquisition which are intrinsic to the role of an expert.

Access to the learning experiences of attempting to solve the problems of a client community is one major source of knowledge in the development of the expertise of an individual expert. The client community provides access to practical experience, a variety of problems, particularly novel ones that go beyond existing expertise, and it also manages the growth of expertise through systems of reward, criticism and access to resources. However, learning from experience is a slow and error-prone process, and socially significant areas of expertise become associated with professional communities that attempt to expedite learning and reduce errors through the sharing of experience. Figure 8 extends the model of Figure 7 to include these other processes, such as apprenticeship, instruction, training, education, workshops, conferences, journals and books. Professional communities usually also play a major role in refining and directing the client community's reward, criticism and resource allocation systems.

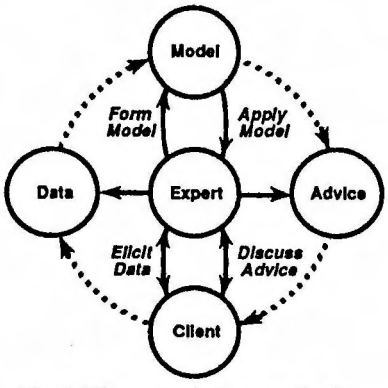


Fig.7 The negotiation cycle in expert-client interaction

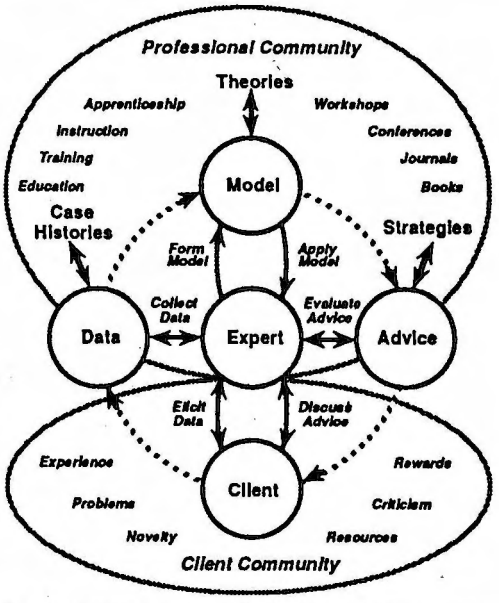


Fig.8 Extended knowledge acquisition process in expert-community interaction

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

ELICITATION FROM MULTIPLE EXPERTS

The knowledge to be acquired for the development of knowledge based systems is often distributed across a group of experts rather than available for elicitation from a single expert . Group elicitation presents major problems because experts can disagree on the use of concepts and vocabulary, and this disagreement may be tacit causing confusion. Methodologies and tools are being developed for knowledge acquisition from groups in which the conceptual frameworks of different experts are compared in a way that makes such disagreements overt and readily identifiable (Shaw & Woodward 1987, Shaw & Gaines 1988). Studies have also been made of extending performance systems based on group elicitation to allow multiple knowledge structures from different experts to be combined without distortion, enabling "dissenting opinions" to be requested (Boose 1986).

Apparent dissent between experts may be due to conflict in using the same terminology for different concepts. In addition apparent difference in viewpoints can result from the use of different terminology for the same concept. These sources of dissent between experts apply equally to the conceptual frameworks of experts and clients. Figures 9 and 10 below illustrate some of the phenomena that can arise by considering the mappings from the concepts of experts or clients to the relevant *distinction* in the domain and the *term* used for that distinction.

In Figure 9: *consensus* arises if the conceptual systems of the two individuals assign the same term to the same distinction; *conflict* arises if they assign the same term to different distinctions. In Figure 10: *correspondence* arises if the conceptual systems of the two individuals assign different terms to the same distinction; *contrast* arises as they assign different terms to different distinctions. *Conflict* shows possible dissent over the application of a concept in a domain since the same term is being used in different ways. *Correspondence* shows possible dissent over the appropriate term to be used for a concept. *Consensus* is particularly important

in showing a shared conceptual sub-system. *Contrast* shows conceptual sub-systems that are not shared but do not lead to dissent.

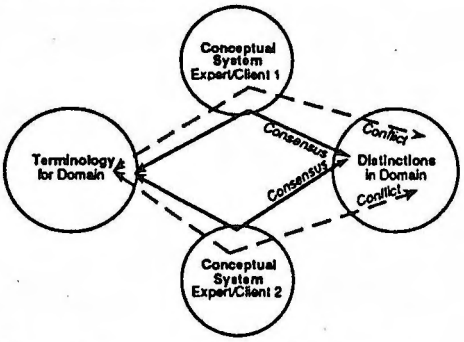


Fig.9 Consensus and conflict between conceptual systems

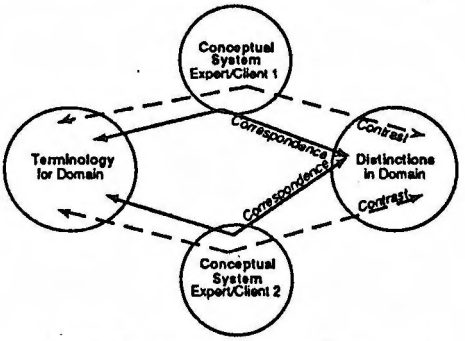


Fig.10 Correspondence and contrast between conceptual systems

Figure 11 is a graphic presentation of a consensus and conflict analysis from a grid exchanged between two experts in geographic mapping techniques. The differences are shown in the grid on the left in numeric form and as shading with darker gray indicating greater difference. The sets of attributes and entities have both been sorted so that those with highest matches indicating greatest consensus are at the top of their respective lists. The graphs on the right show the declining match values and may be used to decide where to place thresholds distinguishing consensus and conflict. Most importantly, this presentation contains sufficient raw and processed data that it may be used for a range of discussions, from simple identification of apparent disagreements, such as the experts disagree on the use of the term "linear interpolation" and the entity "probability matching," to a detailed analysis of how these disagreements have arisen. Measures of correspondence and contrast are presented in a similar graphical form with attributes paired from the two grids.

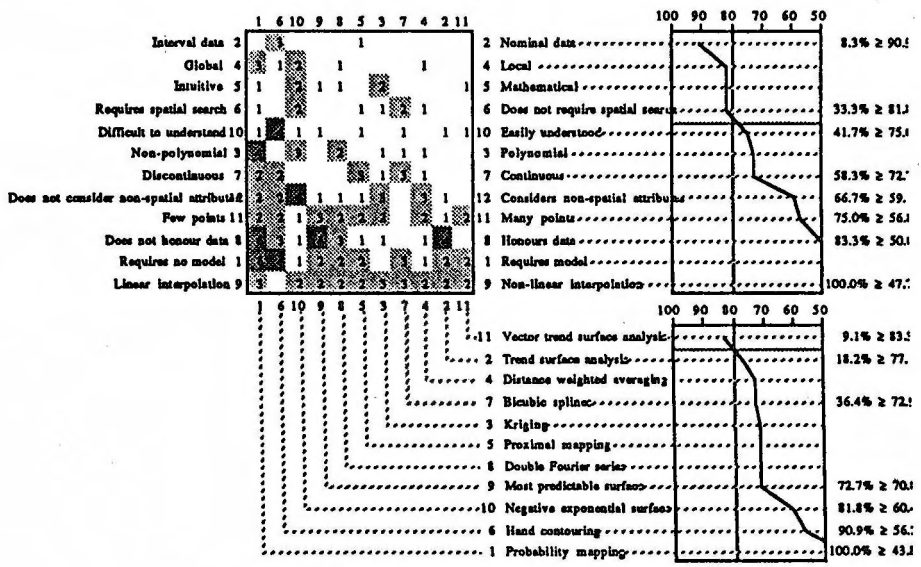


Fig.11 Presentation of measured consensus and conflict

The results of the comparisons are summarized and presented in the four quadrant diagram exemplified in Figure 12. The *consensual* items are simple to deal with—an obvious question is "to what extent can a knowledge-based system be built based only on these well-agreed distinctions?" The *conflicting* items can be looked at with the reverse implication, "to what extent can a knowledge-based system be built not using these ill-defined distinctions?" If some of the distinctions are agreed to be necessary for system development then the basis of the conflict can be examined in greater detail starting with the previous presentation. Many resolutions of the conflict are possible, but the computer-based methodology can go no further than making the basis of conflict and the possible need for resolution apparent.

The *corresponding* items may again be looked at in terms of their relevance to the problem. In addition their further interpretation often leads to significant insights into the knowledge structures involved. For example, the terms *low-level-data—high-level data* and *nominal data—interval or ratio data* corresponding may be interpreted as arising from different *levels of abstraction*. Questions then arise about whether the levels should be represented in the knowledge structures, "which will be best understood by a client?", and "should one term

be used as an explanation of the other in response to a *help* request?" Interpretation of the correspondences in the Figure is an important knowledge acquisition methodology in its own right.

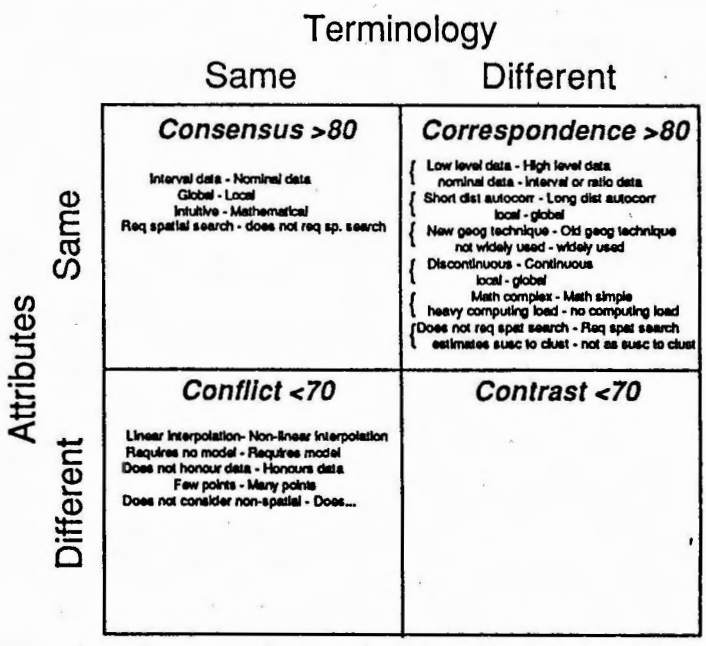


Fig.12 Summary of comparisons between two experts

The simple example shown brings to light abstraction hierarchies, physical and social causal links, and other relations between attributes. Such methodologies have the advantages of group *brainstorming* in stimulating knowledge elicitation, and capturing the results in a highly structured and usable form. Future knowledge acquisition tools will make increasing provision for such inter-expert comparisons.

IMPROVED VALIDATION

The validation of knowledge-based systems is a complex topic in its own right, and a variety of techniques and tools are being developed to support different forms of validation (Shaw & Woodward 1987 Benbasat & Dhaliwal 1988). The objective validation of the knowledge base in terms of its performance, its subjective validation in comparison with experts, and its referential validation in comparison with other knowledge sources should all be supported. Objective validation may require significant case histories that can be collected only from an operational system in actual applications. Referential validation may require deep analysis of derivations of the knowledge base from other knowledge bases. There is a natural separation in subjective validation between experts' direct agreement that their rationale corresponds to that of the knowledge base, and observed correspondence between their behavior and that of the system based on the knowledge base. Both types of validation should be supported.

The validation of knowledge-based systems has all the dimensions of normal scientific validation together with those concerned specifically with the emulation of expertise (Gaines 1988c). As shown in Figure 13, the system can be validated through four primary pathways:

- Through its correspondence to actual events checked against empirical data;
- Through its efficacy in applications checked against actual applications;
- Through the adequacy of its foundations checked against other disciplines corresponding to the derivation of presuppositions from other disciplines—this is sometimes termed 'deep knowledge';
- Through the comprehensibility of explanations given checked against the relevant reference community—corresponding to explanation.

The expert systems paradigm introduces the two additional pathways shown on the right of Figure 13 by taking into account the possibility that there may be "experts" who are competent in applications and explanations:

- Through its correspondence to expert performance checked against the behavior of experts in applications
- Through its correspondence to expert knowledge checked against the rationale of experts in communicating with their reference community

This diagram may be iterated a number of times to show the source of the knowledge structures of experts from empirical data and other disciplines, introducing considerations of the validity of experts and reference communities. Similarly, it may be iterated to show the way in which other disciplines in turn are validated against their own empirical data, deeper foundations, applications and reference communities.

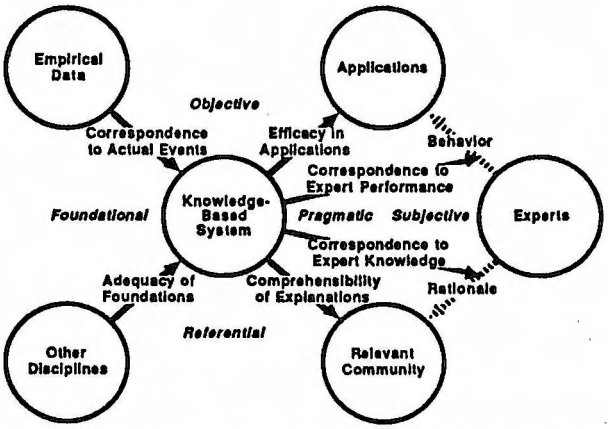


Fig.13 Dimensions of validation of knowledge-based systems

INTEGRATION OF MACHINE LEARNING

This tutorial is primarily concerned with *transfer* of knowledge from existing sources, such as experts and books, rather than *learning* of knowledge from experience. *Machine learning* is an important aspect of knowledge-based systems, but it has a major literature of its own (Michalski & Carbonell 1983, Michalski, Carbonell & Mitchell 1986) which is in large part distinct from issues of knowledge and expertise transfer from existing sources. In a broad sense, learning and transfer processes, are both part of 'knowledge acquisition' for knowledge-based systems, and have many issues in common, particularly those of knowledge representation and inductive modeling. However, most practical knowledge-based systems currently acquire their knowledge base through the transfer of existent knowledge rather than learning from experience, and a major literature has developed about such transfer, its problems, and tools and techniques to overcome them.

Thus, there is currently a major paradigm split in knowledge acquisition research and practice between techniques for the transfer of existing knowledge from human experts and those for the creation of new expertise through machine learning. There is, however, a fundamental relation between the two paradigms in that existing expertise was at some time derived through empirical induction. There is also continuity between the two paradigms in that existing expertise may be partial, erroneous, and of various forms, such that it cannot completely replace empirical induction but may serve to to guide and expedite it. For example human experts in a decision task might know: at one extreme, a complete, minimal set of correct decision rules; through a critical set of cases described in terms of a minimal set of relevant attributes with correct decisions; to, at the other extreme, a source of cases described in terms of attributes among which are those relevant to the decision and with only a greater than chance probability of correct decisions. This is a sequence of decreasing knowledge on the part of the human expert.

Gaines (1989) has studied how the amount of knowledge available from the expert affects the amount of data required for effective empirical induction in the range of situations above. He took a set of cases originally defined by Cendrowska (1987) as a test of empirical induction and generated a range of datasets from it by making incorrect decisions with known probability and adding a prescribed number of irrelevant attributes with random values. He then applied a powerful empirical induction algorithm, INDUCT, to the datasets to determine the average amount of data required to obtain a correctly performing set of rules. The original dataset can be correctly modeled with 6 rules involving 4 attributes.

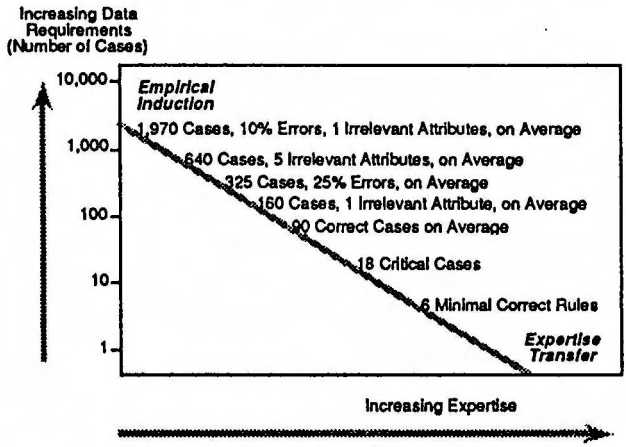


Fig.14 The Trade Off Between Data and Knowledge (Gaines 1989)

Figure 14 shows some of Gaines' results as a plot of the data required on average for empirical induction against the type of knowledge available from the simulated expert. The trade off between data and expertise is clearly shown, and what is also apparent is the continuum between dependence on empirical induction from cases on the left and dependence on transfer of expert knowledge on the right. It is particularly noteworthy that a single algorithm was used for empirical induction across the range of situations Gaines investigated. The results validate the claims of those with tools aimed primarily at eliciting relevant attributes and critical cases from experts that this is an efficient approach. They also show that empirical induction and expertise transfer can be combined to produce a more powerful and versatile acquisition tool than either alone. In future we can expect methodologies and tools to increasingly support a combination of techniques.

INTEGRATION OF TEXT ANALYSIS

A very significant component of human knowledge has already been expressed in text, diagrams and pictures and captured in books and journals. The extraction of knowledge from such material is a major research area in its own right which has major relevance to knowledge acquisition and may be expected to produce methodologies and tools that will be integrated with those described in this tutorial.

One class of research in concerned with the detailed analysis of knowledge structures in text. The CYC project at MCC is a notable example of an attempt to build a knowledge base from an encyclopaedia (Lenat, Prakash & Shepherd 1986). Slator (1988) has described a system for producing text-specific lexicons from selected machine-readable dictionary definitions. Szpakowicz (1988) has described a system that semi-automatically processes technical text and incrementally builds a conceptual model of a domain using Sowa's (1984) conceptual graphs. More simplistic forms of text analysis have been incorporated in a number of knowledge elicitation tools as means of pump-priming the elicitation with a range of entities and concepts. KSSO (Gaines 1987c), KITTEN (Shaw & Gaines 1987), KRITON (Linster 1988) and Cognosys (Woodward 1988) all provide tools for text presentation and semi-automatic analysis as part of the knowledge elicitation process.

As techniques for knowledge extraction from text are developed they will play an increasingly important role in knowledge acquisition. It is unlikely that textual sources will ever replace elicitation from experts for many of the reasons already discussed—knowledge is continually being created and experts, if they write it down at all, do so after considerable delay and with much loss of significant components. However, incorporation of what knowledge is available from written media will become an important source. The same considerations apply to diagrams where, however, the state of the art of knowledge extraction is rather less advanced than with text.

CONVERGENCE WITH HYPERMEDIA

Hypertext and hypermedia are computer-based extensions of existing media such as books and films, and the representation of knowledge in these media has important applications to knowledge acquisition (Nelson 1980, Conklin 1986, Gaines & Vickers 1988). A number of knowledge acquisition tools have already been written in Apple's HyperCard and this trend may be expected to continue. Hypermedia research has focused on the anarchical nature of much knowledge in that it is unstructured and non-algorithmic, where knowledge acquisition research has focused on the ordered nature of some knowledge in that it is well-structured and computer-processible. The term 'knowledge' has come to be used in a very narrow sense in the computing science literature, as 'those aspects of human knowledge that can be formally represented and processed.' This is a major restriction of the usual meaning of knowledge as 'acquaintance with ascertained truths facts or principles'. The philosophical definition of knowledge as 'more than opinion, less than truth' captures the colloquial sense and is a useful concept for system development.

Hypermedia are particularly important in knowledge acquisition because much human knowledge is implicit in action and not formally represented and processed. Hypermedia go beyond knowledge bases in not requiring that the information system be able to represent and process the knowledge encoded in the media used. They subsume knowledge bases but extend knowledge representation to include any media whereby human knowledge can be transferred even if there is no means for analyzing that knowledge or its transfer. For example, a film of a skilled person performing a task which enables another person to perform that task better contains knowledge which can be accessed and transferred through hypermedia (Vickers & Kingston 1987).

Hypermedia are very broadly defined and there is a sense in which they encompass all aspects of knowledge-based systems and their links with other information technologies. Figure 15 attempts to capture the key features of various concepts of hypermedia. Four major conceptual areas are involved: human society and culture, particularly its knowledge processes; instrumentation and control technology, particularly transducers that capture audio-visual events in the world; information technology, particularly databases and information retrieval; and media, ranging from basic drawing materials through to videodiscs.

The *strict hypermedia* definition sees hypermedia as integrating some components of information technology and modern graphic and videodisc media. The growing *hypermedia* concept is more general, seeing them as using information technology to extend all past media. The *generalized hypermedia* definition sees them as *knowledge support systems* (Gaines & Shaw 1987) that integrate existing human knowledge transfer processes with information technology, instrumentation and control technology, and the full range of media. There is no conflict between these interpretations—they differ only in their strictness and are usually differentiated by context.

Developers of knowledge-based systems shells are providing links from them to hypermedia products to support the application of the wide variety of knowledge representations offered. Knowledge acquisition tools are being integrated with hypermedia systems for similar reasons, and this raises interesting possibilities for the representation and use in explanation and tutoring of a much wider range of forms of knowledge than has been possible in the past. The support of knowledge acquisition methodologies through hypermedia tools and the

integration of knowledge acquisition systems, hypermedia systems and knowledge-based systems shells is a major industry trend.

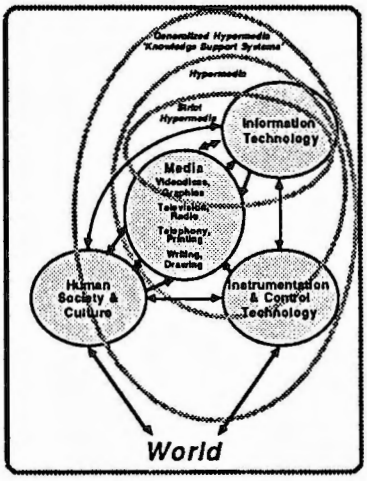


Fig.15 Hypermedia as Knowledge Support Systems

CONVERGENCE WITH CASE

Software engineering for knowledge-based systems development has developed outside the framework of most existing software engineering methodologies. The concepts of "artificial intelligence" and "expert systems" have been taken to have sufficient novelty to justify radically different approaches under the heading of "knowledge acquisition" and "knowledge engineering." However, while some significant differences are apparent, many of the problems of knowledge engineering are ones already addressed in software engineering. Some of the "knowledge acquisition" tools that have been developed are also applicable to the development of decision-support and database systems. However, from another perspective the integration of "knowledge" into information-processing, with the connotations of subjectivity, uncertainty, disagreement and contradiction, introduces new problems of software engineering (Gaines 1988b). We have developed methodologies for "hard" data that do not necessarily transfer easily to the "softer" data that is knowledge.

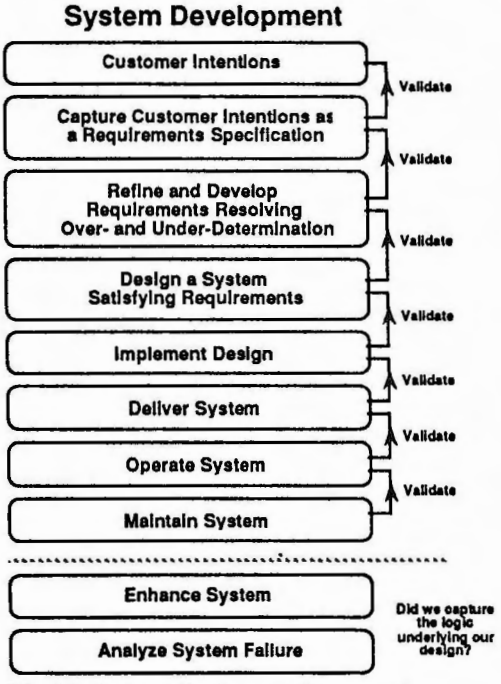


Fig.16 System Design as Supported through Computer-Aided Software Engineering (CASE)

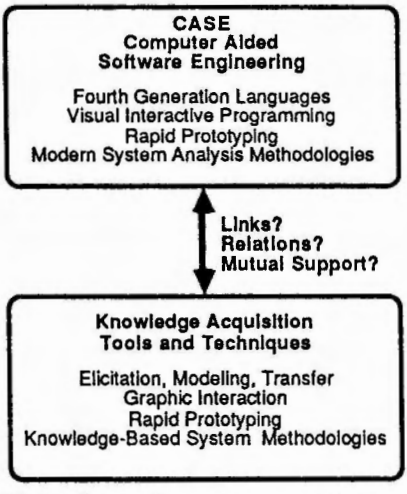


Fig.17 Synergisms between CASE and Knowledge Acquisition

Computer-Aided Software Engineering (CASE) tools range from those that are clearly technical and logical extensions of existing compiler and support environment techniques to those which attempt to deal with many of the "softer" issues of system development such as tracing customer requirements and design decisions (Fisher 1988). Figure 16 shows the many levels of system development for which there now exist structured development methodologies and CASE tools. The diagram is equally applicable to the development of knowledge-based systems, and it is interesting to examine potential applications of CASE tools in KBS development, and the relation between knowledge acquisition tools and similar CASE tools. For example, data design tools using entity-attribute-relation models are related to the entity-attribute, or repertory, grids used in many knowledge elicitation tools.

Figure 17, as a system development model, makes one realize how many aspects are neglected in current knowledge-based system developments. AI technologies are as yet highly immature compared with conventional software engineering. In particular, knowledge base maintenance and system failure analysis will become major problem areas as significant systems come into use. However, knowledge acquisition tools designed to work directly with experts and support them through non-technical interfaces show up very well compared with CASE tools eliciting the same knowledge. The two families of tools have been designed for different markets and user communities.

Since most CASE tools are designed around an open architecture database so that customer tools may be integrated with those supplied, there is tremendous scope for interfacing existing knowledge acquisition tools to CASE databases and using them for preliminary designs of data structures and decision protocols. This is an open area for research and development with exciting practical potential.

CONCLUSIONS

There are many possible approaches to knowledge engineering. However, most researcher and practitioners agree that knowlege acquisition is the bottleneck in the task of developing knowledge based systems. It has been shown in this paper how the role of the knowledge engineer is changing from the repository of expertise to the manager of the whole elicitation and encoding process. Several automated methods already exist, and in this paper some well-established methods of eliciting knowledge used in other disciplines are shown to be relevant to the knowledge acquisition problem. The underlying technological infrastructure was put into the perspective of computer science developments, and related to the social phenonemon of expertise. Some future developments in integrating knowledge acquisition with existing methodologies in other areas were elucidated such as dealing with the problems caused by having to use multiple experts; the problems of validation of knowledge acquisition and knowledge based systems; the use of tools from the field of machine learning; the relationship with hypermedia systems; and the convergence with CASE methodologies.

REFERENCES

Bainbridge, L. (1979). Verbal reports as evidence of the process operator's knowledge. **International Journal of Man-Machine Studies, 11**(4), 411-436 (July).

Bainbridge, L. (1986). Asking questions and accessing knowledge. **Future Computing Systems 1**(2), 143-149.

Benbasat, I. & Dhaliwal, J.S. (1988) A framework for the validation of knowledge acquisition. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Third AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop**. pp.1-1-1-18. Banff (November).

Boose, J. H. (1986) Rapid Acquisition and Combination of Knowledge from Multiple Experts in the Same Domain. **Future Computing Systems 1**(2) 191-216

Breuker, J. & Wielinga, B. (1987) Use of models in the interpretation of verbal data. Kidd, A. (Ed) **Knowledge Elicitation for Expert Systems: A Practical Handbook.** Plenum Press.

Broadbent, D.E., FitzGerald, P. & Broadbent, M.H.P. (1986). Implicit and explicit knowledge in the control of complex systems. **British Journal of Psychology 77**, 33-50.

Cendrowska, J. (1987) An algorithm for inducing modular rules. **International Journal of Man-Machine Studies 27** (4), 349-370 (October).

Collins, H.M. (1985). **Changing Order: Replication and Induction in Scientific Practice.** London: SAGE.

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

Dixon, N. (1981). **Preconscious Processing.** Chichester: Wiley.

Ericsson, K.A. & Simon, H.A. (1984). **Protocol Analysis: Verbal Reports as Data.** Cambridge, Mass. MIT Press.

Fisher, A.S. (1988) **CASE: Using Software Development Tools.** New York: John Wiley.

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

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

Gaines, B.R. (1987c). Rapid prototyping for expert systems. Oliff, M.D., Ed. **Intelligent Manufacturing: Proceedings from First International Conference on Expert Systems and the Leading Edge in Production Planning and Control.** pp.45-73. Menlo Park, California, Benjamin Cummins.

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

Gaines, B.R. (1988b). Software engineering for knowledge-based systems. **Proceedings of CASE'88, Second International Workshop on Computer-Aided Software Engineering.** 14-1-14-7.
Gaines, B.R. (1988c) Knowledge acquisition and technology. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Third AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp.8-1-8-21. Banff (November).
Gaines, B.R. (1989) An Ounce of Knowledge is Worth a Ton of Data: Quantitative Studies of the Trade-Off between Expertise and Data based on Statistically Well-Founded Empirical Induction. KSI-89-2, Knowledge Science Institute, University of Calgary.
Gaines, B.R. & Shaw, M.L.G. (1986). A learning model for forecasting the future of information technology. **Future Computing Systems, 1**(1), 31-69.
Gaines, B.R. & Shaw, M.L.G. (1987). Knowledge support systems. **ACM MCC-University Research Symposium.** Austin, Texas: MCC. pp.47-66.
Gaines, B.R. & Vickers, J.N. (1988). Design considerations for hypermedia systems. **Microcomputers for Information Management,** 5(1), 1-27 (March).
van Gundy, A.B. (1981). **Techniques of Structured Problem Solving,** Van Nostrand Reihold:New York.
Hagstrom, W.O. (1965). **The Scientific Community.** New York: Basic Books.
Hawkins, D. (1983). An analysis of expert thinking. **International Journal of Man-Machine Studies,** 18(1), 1-47 (January).
Johnson L. & Johnson, N.E. (1987) Knowledge elicitation involving teachback interviewing. Kidd, A. (Ed) **Knowledge Elicitation for Expert Systems: A Practical Handbook.** Plenum Press.
Johnson, P.E., Zualkernan & Garber, S. (1987). Specification of expertise: **International Journal of Man-Machine Studies,** 27(2), 161-181 (February).
LaFrance, M. (1987) The knowledge acquisition grid: a method for training knowledge engineers. **International Journal of Man-Machine Studies,** 27(2), 245-255 (February).
Lenat, D., Prakash, M. & Shepherd, M. (1986). CYC: Using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks. **AI Magazine** 6(4), 65-85.
Linster, M. (1988) A critical look at KRITON. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Third AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp.17-1-17-19. Banff (November).
McGraw, K.L., and Harbison-Briggs, K. (1989). **Knowledge Acquisition Principles and Guidelines,** Prentice Hall: New Jersey.
McKeown, B., and Thomas, D. (1988). **Q Methodology** Sage California.
Michalski, R.S. & Carbonell, J.G., Eds. (1983). **Machine Learning: An Artificial Intelligence Approach.** Palo Alto, California: Tioga.
Michalski, R.S., Carbonell, J.G. & Mitchell, T.M., Eds. (1986). **Machine Learning: An Artificial Intelligence Approach Volume II.** Los Altos, California: Morgan Kaufmann.
Nelson, T. (1980). Interactive systems and the design of virtuality. **Creative Computing,** 6(11), 56-62 (November).
Nisbett, R.E. & Wilson, T.D.. (1977). Telling more than we can know: verbal reports on mental processes. **Psychological Review 84,** 231-259.
Pask, G. (1975). **Conversation, Cognition and Learning,** Elsevier, Amsterdam.
Pidgeon, N.F., Turner, B.A. and Blockley, D.I. (1989) The Use of Grounded Theory for Conceptual Analysis in Knowledge Elicitation, **International Journal of Man-Machine Studies,** (in press).
Rusk, R.A. & Krone, R.M. (1984) The Crawford Slip Method (CSM) as a tool for extraction of expert knowledge, Salvendy, G. (Ed.) **Human-Computer Interaction,**. Elsevier, New York, pp279-282.
Shaw, M.L.G. & Gaines, B.R. (1987) KITTEN: Knowledge Initiation & Transfer Tools for Experts & Novices. **International Journal of Man-Machine Studies** 27(3), 251-280 (September).
Shaw, M.L.G. & Gaines, B.R. (1988) A methodology for recognizing conflict, correspondance, consensus and contrast in a knowledge acquisition system. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Third AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp.30-1-30-19. Banff (November).
Shaw, M.L.G. & Woodward, J.B. (1987) Validation of a knowledge support system. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Second AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp.18-0-18-15. Banff (October).
Slator, B.M. (1988) Extracting lexical knowledge from dictionary text. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Third AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp.33-1-33-18. Banff (November).
Sowa, J. (1984) **Conceptual Structures.** Addison Wesley Publishing Co. Reading Ma. 1984.
Steels, L. (1986). Second generation expert systems. Bramer, M.A. (Ed). **Research and Development in Expert Systems III.** pp.175-183. Cambridge: University Press.
Szpakowicz, S. (1988) Semi-automatic acquisition of conceptual structure from technical texts. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Third AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp.35-1-35-16. Banff (November).
Vickers, J.N. & Kingston, G.E. (1987). Modelling the master coach: building an expert system for coaching. Norrie, D., Ed. **Learning in Future Education: Proceedings of International Conference on Computer Assisted Learning in Post-Secondary Education.** pp.207-212. University of Calgary.
Woodward, B. (1988) Knowledge engineering at the front-end: defining the domain. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Third AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp.37-1-37-25. Banff (November).

The Problem of Knowledge Extraction with Human Experts: A Case of Expert Verbalizations.

M. Haers* & G. d'Ydewalle

Laboratory for Experimental Psychology
University of Leuven
Tiensestraat 102,
B-3000 LEUVEN
BELGIUM

ABSTRACT

In this paper we present two experiments in which the relationship between expertise and verbalizable knowledge is studied. Four verbalization techniques are evaluated in the context of knowledge extraction with human experts: thinking aloud, questionnaire answering, retrospective verbalizations and expert-novice communication.

INTRODUCTION

When building an expert system, one of the tasks of the knowledge engineer is to elicit task related knowledge from the expert that is necessary for construction and validation of the knowledge base and for explanation and tutoring purposes. The problem of eliciting the knowledge from human experts is often called the 'Bottleneck' in building expert systems. Expert knowledge is compiled, automatized and consequently inexplicit. Furthermore, it seems that the explanations given by an expert, when asked to verbalize his/her heuristic knowledge, are inconsistent, incomplete and even possibly incorrect (Glaser, Chi, & Farr, 1988). Moreover, there exist strong time limitations and motivational problems when working with a human expert. Thus, it seems necessary to study the problem of knowledge extraction with human experts more carefully and to develop efficient techniques for knowledge elicitation.

Until now, a large number of knowledge extraction tools and techniques have already been developed to assist the knowledge engineer in extracting the expert knowledge (Boose, 1988). However, most of these tools are mainly applicable in very particular domains of expertise, often limited only to the specific domain where it was built for. Moreover, only few studies report on the validation of the tools for knowledge extraction in terms of the reliability of the data, the generality of the use of the tool, etc. (Shaw, 1988). At the Second EKAW88, Boose (1988) proposed a framework for classification and evaluation of knowledge acquisition tools that can be used in this context.

---

* This author is sponsored by the Belgian I.W.O.N.L./I.R.S.I.A.

The problem of knowledge extraction with human experts is challenging for psychological research (Hoffman, 1987). Experimental psychology not only offers methodologies for controlled research on expert knowledge, it also offers a theoretical framework from which the knowledge extraction research can profit. Interesting theoretical questions in this domain concern the nature of implicit knowledge, the development of human expertise, human learning and problem solving, etc. More applied questions concern the development of techniques for the elicitation of human expertise, that are based on psychological insights concerning the explicitness of human expert knowledge. In the study of human problem solving for example, the technique of protocol analysis of simultaneous and retrospective verbalizations of expert problem solvers have been widely used (Newell & Simon, 1972; Ericsson & Simon, 1980,1984). Furthermore, multidimensional scaling techniques have been used to estimate the similarity or non-similarity between different concepts within the knowledge base (Cooke & McDonald, 1987), in combination with the repertory grid technique (Boose, 1985). As a result, network structures can be described. A third technique concerns the use of interviews (structured and unstructured) for knowledge elicitation purposes. Sorting cards or concepts can help to organize or to structure the knowledge in the knowledge base (Burton, Shadbolt, Rugg, & Hedgecock, 1988). There are still other techniques that can help the knowledge engineer either to fill in the knowledge base or to organize it. Studying this kind of techniques from a psychological point of view gives the opportunity to evaluate the way of extracting the expert knowledge on psychological grounds and to determine by this the psychological plausibility of the data.

The present research deals with the study of the development of techniques for knowledge elicitation, based on psychological insights. In what follows we present two studies that focus on the use of verbalization techniques for knowledge elicitation purposes as well as on factors that facilitate the verbalization of knowledge that is associated with the task performance of an expert.

For both experiments we use the experimental paradigm of Broadbent and others (Berry & Broadbent, 1984; Broadbent & Aston, 1978; Broadbent, Fitzgerald & Broadbent, 1986), that has been developed to study the relationship between performance on a cognitive decision task and associated verbalizable knowledge. The procedure of the paradigm consists in two phases: In a first phase subjects learn to control a dynamic system with two variables, that is simulated on a computer. The task consists in obtaining and maintaining as long as possible a target output value by choosing for each consecutive trial the value of the input parameter. By working on this task subjects are asked to think out loudly in order to be able to record their simultaneous verbalizations. The first phase results in a task performance score that corresponds with subjects' level of expertise. In a second phase then, subjects' verbalizable knowledge is measured by means of a questionnaire and two general questions, that retrospectively ask for the functioning of the system and for the way they handled the task.

The main results of this series of studies can be summarized as follows. There exists a dissociation between task performance and associated verbalizable knowledge as measured by a questionnaire: Subjects obtain higher task scores due to more task experience, without a comparable enhancement in the questionnaire scores. Furthermore, detailed verbal instructions before starting the task leads to better questionnaire scores, without affecting the task scores. Finally, the combination of verbal instructions and thinking aloud makes subjects perform better on the task without affecting the questionnaire scores any further. These results made Berry and Broadbent (1984) to conclude that the control of the dynamic

system is learned in an implicit way that makes verbal explicitation further impossible.

In our first experiment we replicate the study of Berry and Broadbent (1984) to investigate the relationship between task performance (= expertise) and associated verbalizable knowledge as a means to evaluate the use of simultaneous and retrospective verbalizations of human experts for knowledge extraction purposes. Therefore, subjects are first trained to control a dynamic system in a certain amount of trials. Simultaneously, several manipulations are meant to facilitate subjects' verbalization abilities. In Experiment 2, the verbal interaction between a well trained subject and a novice is recorded and analyzed. The results of the two experiments are interpreted in terms of the use of human experts' verbalizations for knowledge elicitation purposes.

Experiment 1

Following Berry and Broadbent (1984), subjects are asked to work on a cognitive decision with two variables. The task consists in the management of a sugar factory. Subjects have to control the output level of sugar production by giving in on each consecutive trial the (input) number of workers they want to employ. Diverse manipulations all are meant to enhance subjects' verbalizable capacities. First, half of the subjects are asked to think aloud while working on the task. With this manipulation we hope subjects will be more able to give some verbal comments on their way of solving the task after termination. Furthermore, by analyzing these thinking aloud protocols thoroughly (cfr. Ericsson & Simon, 1980, 1984), we hope we can understand better the way the subjects learn to control the task (development of expertise). A second manipulation that is meant to facilitate subjects' task related verbalizations consists in the administration of a questionnaire at two occasions during the experiment. This questionnaire is considered as a measurement of subjects' task related verbalizable knowledge because answering it correctly assumes that the subjects have built up some verbal knowledge concerning the task. Third, following each administration of a questionnaire, the subjects are asked to answer two general questions (retrospective verbalization) concerning the way they solved the task. These two questions are meant to cue the subjects to verbalize their task related performance. Finally, the fourth manipulation to enhance the verbalization facilities of the subjects consists in providing half of the subjects with detailed verbal instructions before starting the task. These verbal instructions contain more detailed information about the way work force and level of production are interrelated. The verbal instructions are meant to prompt the subjects to think more verbally about the task and to obtain better results on the verbalization measurements.

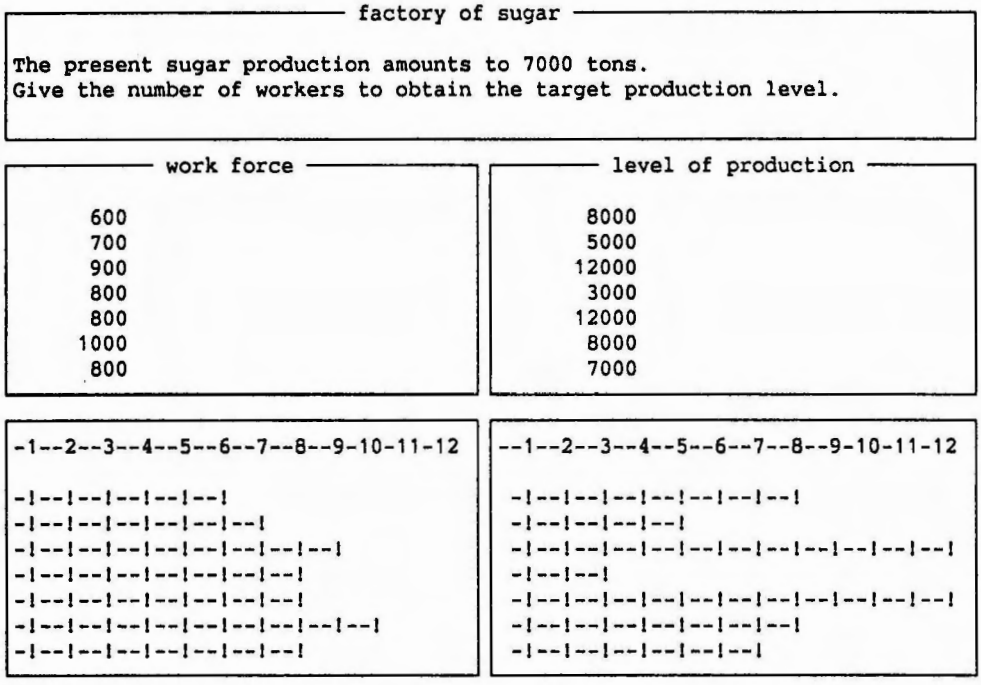
METHOD

The subjects and the task.

Twenty four subjects (12 male and 12 female), aged between 19 and 24 are divided between the eight conditions (cfr.infra). They take place in front of a PC and are instructed to imagine that they are in charge of a sugar production factory. They have to continually control the sugar production by deciding on each trial how many workers they want to employ. The size of the work force and the level of production are both discrete variables, each with 12 possible values. The size of work force is expressed in hundreds of workers, going from 100 to 1.200 workers. The level of

production can vary between 1.000 and 12.000 tons of sugar (see Figure 1).

Figure 1: An example of the task. This information is given to the subjects during task performance.



The task starts with the initial situation of 6.000 tons and 600 workers. The subjects have to type in a number of workers they want to employ on each consecutive trial, after which the computer calculates the corresponding production level. The resulting production level constitutes the new situation in the factory for which the subjects have to type in a new number of workers, and so on. Work force and sugar output are scrolled on the screen, numerically as well as graphically (see Figure 1). The computer calculates the corresponding production level by applying the following formula:

$$P[new] = (20 \times W) - P[old]$$

(W is a number between 1 and 12, representing the number of hundreds of workers typed in by the subject; P[new] is the resulting level of sugar production expressed in thousands of tons and P[old] is the production level on the previous trial.)

Before showing the resulting production level on the screen, the computer adds, on a random basis 1, 0 or -1 to the calculated production level. The subjects are informed neither about the formula nor about the manipulation of the random element.

The task consists in trying to reach and maintain a target output level of 9000 tons of sugar as much as possible in the consecutive trials. The subjects receive three sets of 30 trials, alternated with the manipulations

according to the condition they are in. Each correct answer is scored with one point, with a correction for the random element. The maximum score amounts to 30 points per set. We consider a score of 70% correct answers as being on expert level.

The questionnaire and the two general questions.

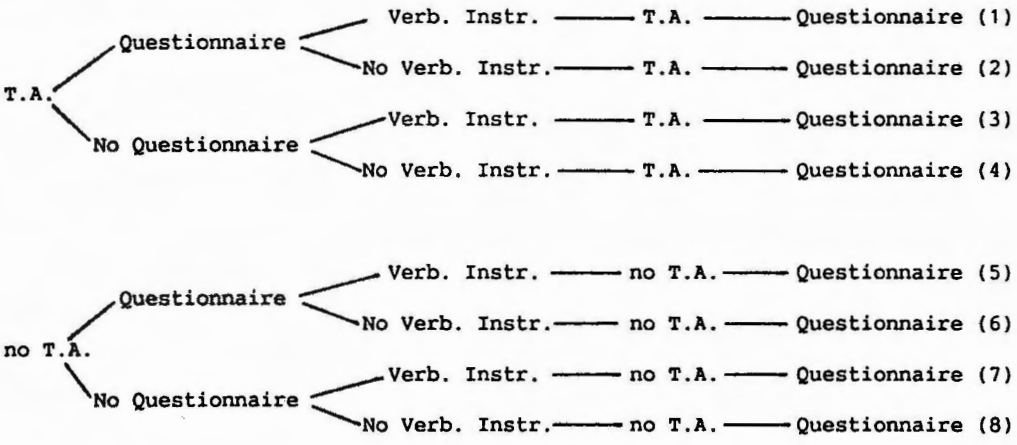
According to their condition, subjects receive one or two questionnaires to answer aloud while typing in the answers on the PC. The questionnaires consist of eight questions, divided into four sections. Each question deals with a possible situation in the sugar factory for which subjects are asked to make a decision according to either the missing number of work force and/or the missing level of sugar production.

All subjects receive two general questions at two occasions during the experiment: after they finished the first set of trials, and at the very end of the experiment. Both questions retrospectively ask for general information about the way they solved the task. The questions are: "Can you explain how you did proceed in performing the task?" and: "Can you tell how you think work force and sugar production are interrelated?".

Design and procedure.

The design is schematically presented in figure 2.

Figure 2: Schema of the design and the procedure in experiment 1.



Half of the subjects is asked to think aloud (T.A.) while performing the task during the three sets of 30 trials. They are asked to give a reason for each number of workers they type in. For the other subjects no specific instruction is given. Orthogonal to this manipulation, half of the subjects is given a first questionnaire after having completed the first set of trials, followed by the two general questions. The other half of the subjects only receives the two general questions at that time. Finally, half of the subjects (again orthogonally) receives detailed verbal instruction (verb. instr.) before starting the second set of 30 trials while the other subjects do not. After having performed the three sets of trials each subject has to perform a questionnaire again followed by the two general questions. This results in 2 x 2 x 2 factorial design. Each subject is

randomly assigned to one condition.

DATA ANALYSIS

1. Task performance scores.

The task performance scores are not very high: Mean performance scores on the task are 17.6% for the first set, 35.6% for the second set and 40.5% for the third set.

A (2 x 2 x 2 x 2 . 3) analysis of variance is performed on the task performance scores with thinking aloud versus no thinking aloud, verbal instruction versus no verbal instruction, number of questionnaires (one or two) and sex as independent variables between-subjects and level of task experience (the first, the second or the third set of trials) as independent variable within-subjects. A significant main effect is found of level of experience [$F(2,16)=14.33$, $p<0.001$]. The significant main effect of task experience means that the more experience the subjects have on the task, the better their task performance scores will be. These results were also reported by Berry and Broadbent (1984). Although there is an improvement in the task performance, the overall scores are not very high.

Second, a significant interaction effect between thinking aloud versus no thinking aloud, number of questionnaires, verbal instruction versus no verbal instruction and level of experience is found [$F(2,16)=4.515$, $p<0.05$]: The highest task performance score is obtained when combining the manipulations of thinking aloud, verbal instruction and the administration of two questionnaires. There is no clear effect of one of the manipulations separately. These results are similar to those reported by Berry and Broadbent (1984).

2. Questionnaire scores.

The mean scores for the questionnaires are 27.8% for the first (i.e. after 30 trials) and 43.8% for the second (i.e. after 90 trials) administration of questionnaires, respectively.

Two analyses of variance are performed on the questionnaire scores. No significant effects are found. With these results we could not replicate Berry and Broadbent's (1984) results, concerning the facilitating effect of the different manipulations on the questionnaire score. Our results thus indicate poor verbalization possibilities of the subjects as measured by the questionnaire, which are not influenced by the manipulations.

3. Relation between task performance and verbalization scores.

We found no relation between task performance and verbalization neither after 30 trials nor after 90 trials. This lack of correlation illustrates the dissociation between task performance and verbalization as was also found by Berry and Broadbent (1984).

PROTOCOL ANALYSIS

Based on the work of Ericsson and Simon (1980,1984) for investigating verbal protocols, we analyzed the verbalizations of the subjects during task performance and during questionnaire answering.

In summary, the most important conclusions of these protocol analyses are the following:

1. Concurrent verbalizations during task performance.

Subjects seem to adopt different strategies, by which we mean well defined ways of working on the task (e.g. trial and error; testing for one to one correspondence between number of work force and production level; trying out mathematical formulae). These strategies change as a function of training. Due to the lack of enough experience with the task, it was not possible to study in more detail the intra-individual development of expertise.

2. Concurrent verbalizations during questionnaire answering.

This analysis reveals that subjects answer the questionnaire in a totally different way as they solved the task: The questions are answered in a very superficial way without taking into account the knowledge gathered during task performance. This is even true for the subjects who answered two questionnaires: Once subjects have verbalized their task related knowledge, by answering the first questionnaire, they use the same formulations at a next occasion even if it is inefficient. This again confirms the dissociation between task performance and verbalizable knowledge. However, interpreting the different scores on the task and on the questionnaire only in terms of this dissociation is not sufficient. Probably, the questionnaire is not built in the right way as to directly tap into subjects' verbalizable knowledge, what could lead to dissociated scores. We therefore change the format of the questions in Experiment 2 as to better resemble the task situations.

3. The analysis of the two general questions.

The protocol analysis of the two general questions (retrospective verbalizations) reveals that subjects are less complete and less detailed than they are in the thinking aloud protocols (simultaneous verbalizations). This finding is important for our research question, in that making experts verbalize simultaneously leads to more accurate protocols, providing that the verbalization act does not interfere with the task under study (Ericsson & Simon, 1980).

CONCLUSION

Our conclusion for this experiment is, analogous to Berry and Broadbent's (1984), that there exists a dissociation between task performance and related verbalizations as measured by a questionnaire, at least when one starts learning the domain task. Maybe this last comment will need some explanation. As is stated earlier, we only gave the subjects three sets of 30 trials to work on. No subject reaches the expert level at the end of the experiment. For that reason we cannot make any conclusions about the relation between (expert) task performance and related verbalizations. We therefore thought it would be interesting to give subjects the possibility to work on the task for some more sets of 30 trials, to make them more experienced with it (see Experiment 2). Furthermore, as is shown in Experiment 1, subjects work differently on the task and on the questionnaire: Both are perceived as totally different tasks. Therefore, in Experiment 2, the question situations are adapted as to better resemble the task situations. Finally, as the questionnaire scores are not influenced by the experimental manipulations (verbal instruction, the administration of one or two questionnaires), we will not take these factors further into account. The verbal reports of the subjects neither seemed influenced by the thinking aloud instruction, so that thinking aloud will further be applied only to follow the development of subjects' expertise.

Experiment 2

The aim of Experiment 2 is to look for other techniques or at least for other manipulations to facilitate one's verbalization abilities. As no subject became an expert in the first experiment, the subjects are now asked to work on ten consecutive sets of 30 trials, instead of only three sets, with continually thinking aloud. After this, the subjects are asked to explain their expertise to a naive novice. The obtained communication protocols are then analyzed and evaluated in the context of knowledge extraction with human experts: How does the trained subject structure this conversation; what information is made explicit and what is left inexplicit; what about the transfer of expertise and how useful is the communication between a trained subject and a novice for knowledge elicitation purposes?

In this study we will be able to evaluate four techniques for knowledge extraction: The thinking aloud protocols of the subjects during their training period, the administration of the post-task questionnaire that resembles more the task situations, the two general questions asking retrospectively for the way the subjects worked on the task, and the communication protocols between trained subjects and novices.

METHOD

The subjects and the task.

12 female and 12 male subjects are equally divided into three conditions (see Figure 3). Not one of the subjects did already participate in the previous experiment. The task is not different from the dynamic cognitive decision task in Experiment 1.

Two general questions and the questionnaire.

After the subjects have finished to work on the task, they answer two general questions and a questionnaire, on paper. No thinking aloud is recorded. The general questions retrospectively ask for information about the way the subjects have been working on the task. The questions immediately follow the task performance to reduce memory effects. Then a written questionnaire is given. This questionnaire is the same as in Experiment 1, except that care is taken to make the questions more look like the task situations.

Design and procedure.

There are three conditions with eight subjects randomly assigned to each of them (see Figure 3). Before starting with the task, the experimenter gives a short standardized description of the task and the way to work on the PC by means of an example set of trials. This explanation is the same for all subjects. In the trained-subjects condition (T) the subjects then work on the task for ten consecutive sets of 30 trials. The experimenter asks them to think out loudly (by giving a reason for each answer they type in) while performing on the task. These thinking aloud protocols are tape recorded and transcribed verbatim afterwards. After having gone trough the ten sets of trials, they are asked to explain to a novice the working of the system and their own way of controlling the system. After this communication phase the subjects work on two test sets of 30 trials. Finally, the two general questions concerning the way of working on the task followed by the questionnaire are presented on paper. In the novice-communication condition (Nc), each subject meets a T-subject who explains how the system works and how he/she has been working on the task, immediately after the standard instructions of the experimenter. After the communication phase, the Nc-subjects are asked to work on two successive test sets and to answer on

paper the two general questions and the questionnaire. Finally, in the novice condition (N) the subjects start to work on two sets of trials without thinking aloud, after a short explanation by the experimenter. They finish the experiment with answering of the two general questions and the questionnaire.

Figure 3: Schema of the design and the procedure of experiment 2.

Trained (T) subjects (n=8)	Novices-communication (Nc) (n=8)	Novices (N) (n=8)
Brief instructions by the experimenter. Demo of the task by means of a standardized example.	Brief instructions by the experimenter. Demo of the task by means of a standardized example.	Brief instructions by the experimenter. Demo of the task by means of a standardized example.
↓	↓	↓
Training phase: ten sets of 30 trials	No training phase	No training phase
↓	↓	↓
Communication phase: T-subject explains the task to the Nc	Communication phase: Nc-subject receives explanation from T	No communication phase
↓	↓	↓
Two test sets of 30 trials	Two test sets of 30 trials	Two test sets of 30 trials
↓	↓	↓
Two general questions	Two general questions	Two general questions
↓	↓	↓
Questionnaire	Questionnaire	Questionnaire

DATA ANALYSIS

1. Task performance scores.

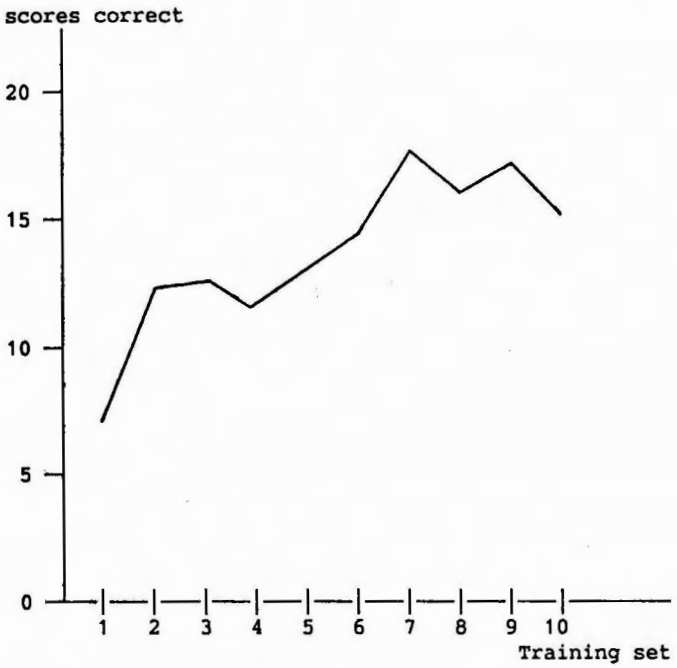
A trend analysis is performed on the task performance scores of the first ten sets of the T-subjects to investigate how task performance develops with more experience. A significant linear trend is found [$F(11,77) = 5.45$, $p < 0.00001$] (see Figure 4).

A (2 x 3 . 2 x 2) analysis of variance is performed on the task performance scores with sex and condition as independent between-subjects variables and first or second set and first or second part of the set as independent within-subjects variables. These results show a significant main effect of condition [$F(2,18)=5.066$, $p<0.02$]. The mean percentages correct answers are 55%, 37% and 18% for the T-subjects, Nc-subjects and N-subjects respectively. An a posteriori Tukey test shows that there is a significant difference between the scores of the T-subjects and the N-subjects ($p<0.05$). Apparently, due to the long training and the availability of some form of more explicit knowledge, subjects perform significantly better on the task than in the absence of both variables. However, it is important to note that only two of the T-subjects can be considered as 'genuine' experts (see Figure 5). By this, the most important results will be limited to these two experts.

The difference in task performance scores between Nc-subjects and the N-subjects, is not large enough to reach statistical significance, but is large enough to make the difference between T-subjects and Nc-subjects no longer significant. We will come back to these differences after we

discussed the relation between the scores of the T-subjects and their Nc-partners.

Figure 4: Task performance scores of the T-subjects as a function of task experience. On the X-as the subsequent ten sets of trials are given. The values on the Y-as correspond with to the mean task performance scores (max. score = 30).

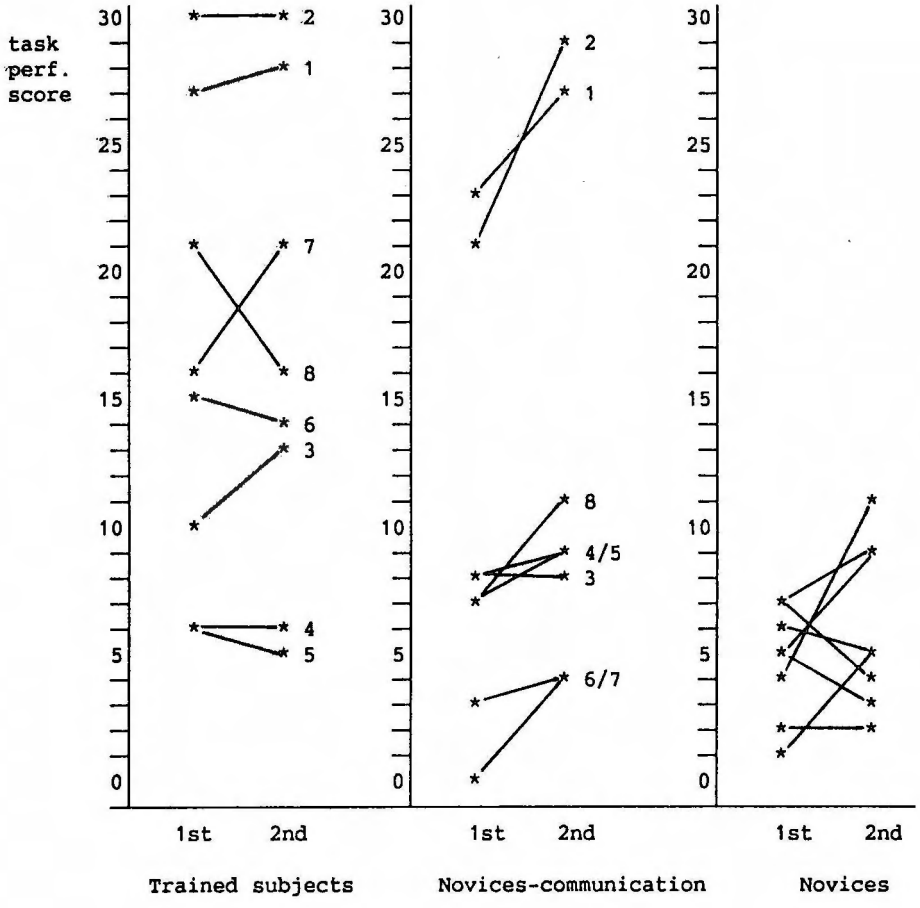


The main effect of the sets is also significant [F(1,18)= 5.649, p<0.03], with higher performance scores on the second set of 30 trials than on the first set (39% vs. 34% respectively). Apparently, subjects' task score increases with more experience, irrespective of condition since there is no interaction effect between set and condition.

We further want to investigate the transfer of task related knowledge due to the communication phase between T-subjects and Nc-subjects. The product-moment coefficient only reaches significance when measured between task performance scores of the T-subjects and their Nc-partners in the second test set: The task performance scores of the Nc-subjects are high or low according to the scores of the T-subjects from whom they received verbal explanation, only in the second test set. Moreover, when considering these data in more detail (see Figure 5), some interesting findings emerge. Due to the fact that only two of the T-subjects did obtain 'expert' levels of performance, an additional analysis of variance was performed on the task performance scores of the T-subjects and the Nc-subjects, with as a between-subject factor the fact whether the Nc-subject was instructed by an 'expert' T-subject or not and, with as a within-subject factor first versus second set. The former factor turned out to have a significant main effect [F(1,6) = 70.006, p<.001]. Thus, communication by a T-subject has a beneficial effect on the performance of the Nc-subject, provided that the former can be

considered as a genuine 'expert'. Second, first versus second set had a significant main effect [F(1,6) = 26.182, p<.01]. All subjects perform better on the second set than on the first set. This explains why only the coefficient between the scores of the T-subjects and their Nc-partners for the second set is significant (cfr. supra). Moreover, the interaction between the two factors is significant [F(1,6) = 6.545, p<.05]: Task scores of the Nc-subjects who receive communication from 'experts' obtain higher improvements in their scores from the first set to the second set than the other Nc-subjects. This is an important finding in the context of our research question: The usefulness of communication with a novice as a technique for knowledge elicitation. Not only do subjects, who receive communication from an 'expert', perform better than other subjects, the former also benefit more from experience after the communication phase.

Figure 5: Task performance scores of the subjects in the first and the second test sets. The numbers refer to the corresponding communication partners. For example: The scores for the novice-communication condition that are indicated with number 1 refer to the performance scores of Nc-subject number 1, who received verbal explanation from trained subject number 1.



2. Questionnaire scores.

The mean questionnaire scores of the subjects are 34%, 44% and 61% respectively for the N-, the Nc- and the T-subjects. A (2 x 3 . 4) analysis of variance on the questionnaire scores with sex and condition as independent between-subjects variables and the scores on each section of the questionnaire as dependent variable does not show significant effects. It seems that the questionnaire scores cannot differentiate between the three conditions.

We further want to know how well the scores on the questionnaire for both the T-subjects and the Nc-subjects are correlated. The product-moment coefficient amounts to $r=0.69$, without reaching significance.

Similarly as for the task performance scores, a separate analysis on the questionnaire scores of the Nc-subjects is performed, with as a between-subject variable the fact whether the Nc-subjects received communication from an 'expert' T-subject or not. The results show a main effect of 'expert' level of the communication partner [$F(1,6) = 14.40$, $p<.0.01$]. The mean percentage correct answers on the questionnaire amounts to 81% for the Nc-subjects who got detailed explanation from an 'expert' T-subject and 31% for the other Nc-subjects. This means that Nc-subjects, who are informed by an 'expert' T-subject, perform better on the questionnaire, than when their communication partner does not perform at expert level. The correspondence between the questionnaire scores of the expert T-subjects and the Nc-subjects is not perfect however. As will be seen further in the analysis of the verbal commentaries on the questionnaire, expert T-subjects use different strategies to answer the questionnaire than the Nc-subjects.

3. Correlation between task performance and questionnaire scores.

We finally calculated the correlation between task scores and questionnaire scores. The overall product-moment coefficient is 0.72 ($p<0.005$). Product-moment coefficients of task performance and performance on the questionnaire for the T-subjects, the Nc-subjects and the N-subjects respectively are 0.67, 0.77 ($p<0.05$) and -0.13. The higher coefficients for the T-subjects and for the Nc-subjects as compared to the N-subjects can be the result of experience and/or explicit verbal information. For those subjects who get more experience with the task and/or who get some explicit verbal instruction, the performance scores on the task and on the questionnaire tend to be more similar than for the subjects with no experience nor explanations at all.

PROTOCOL ANALYSIS

As is stated earlier, only two of the T-subjects reached a performance level that can be considered at expert level. We therefore did limit the number of protocols to analyze, to these two subjects. In what follows, a brief summary of the results of the protocol analyses (thinking aloud protocols, communication protocols, questionnaires and retrospective general questions) will be given.

1. Analysis of the thinking aloud protocols.

The thinking aloud protocols of the two 'experts' indicate the development of production rules that become more specific with more experience. These subjects succeed in finding out a production rule by which they can control the system very well. The verbal reports of the experts become more succinct with more experience: A detailed description of the way they work is replaced by the mere application of a self-formed rule.

2. Analysis of the communication protocols.

The analysis of the communication protocols reveals a structured way of explaining the control of the system to the Nc-subject, without going too much in detail. 'Expert' T-subjects seem able to explain their expertise to a novice in an efficient and clear way.

3. Analysis of the general questions and the questionnaire.

This analysis concerns the written commentaries following answering the questions. There is no difference between the answers of the T-subjects and their communication partners on the general questions. The Nc-subjects roughly take over the same explicit verbalizations as they heard during the communication phase.

The analysis of the questionnaire answering reveals a less flexible way to answer the questions by the Nc-subjects as compared with their expert T-communication partners. Especially for answering the questionnaire, more specific knowledge seems necessary that is not transferred through the communication phase. From a more theoretical point of view we tried to interpret these differences in availability of task related knowledge of T-subjects versus Nc-subjects in terms of implicit (as a result of training) versus explicit (verbalizable) knowledge (Haers, 1988).

CONCLUSION

The results of this second experiment also indicate a dissociation between expertise in the dynamic decision task and verbalizable knowledge as is measured by a questionnaire. However this dissociation is much less pronounced due to the longer training of the subjects and due to the fact that the questionnaire is more adapted to the task situations. This last point is interesting in the context of knowledge extraction with human experts: In order to be useful for knowledge extraction, questionnaires must be made more look like the task situation under study.

The main aim of this experiment is to evaluate different verbalization techniques as methods for knowledge extraction. From this experiment it becomes clear then that the conversation between an 'expert' and a novice, concerning how to solve the decision task, is a promising technique for extracting task related knowledge. An expert seems able to explain rather clearly about his expertise so that a naive subject can take over his way of solving the decision problem and end up with an expert task score. By observing these communication protocols carefully, we can analyze more in detail what a conversation of an expert with a novice looks like, what is made explicit to a novice (and what not) and how complete 'experts' are in their explanations. In combination with thinking aloud and with the answering of a specific questionnaire and general questions, this communication phase seems worth-while.

DISCUSSION

The two experiments reported here describe the use of concurrent and retrospective verbalizations as knowledge elicitation techniques. We trained subjects under controlled conditions to become expert in a cognitive decision task. The analysis of the task-related verbalizable knowledge was performed in different ways. The results show that verbal instruction of a novice by a well trained subject enables the former to perform on the same expert level as the latter. This result at least indicates the importance of

this kind of verbal explanation when conducting a knowledge extraction phase. However our research data also show that using this communication phase only is not sufficient: After this communication phase the novices seem not able to answer a task related questionnaire correctly. This still points to an inexplicit part of experts' knowledge.

Furthermore, we can state that - in the light of the research question - task performance and associated verbalizable knowledge, as measured by the questionnaire, are positively correlated when subjects become more experienced in the task domain. This is an important finding for the knowledge extraction research when using verbalizations of experts as an elicitation technique. Verbalizations yield interesting information. The problem is to know how to use it and how to analyze the data. Our experimental research at least offers some guidelines for the use of verbalizations. First, simultaneous verbalizations yield more detailed and complete information than retrospective verbalizations. It seems thus more useful to ask subjects to reason out loudly than to give them some retrospective questions to answer afterwards. Answering retrospectively leads to more general and rationalized reports, that often do not correspond with reality and that are more affected by previously given verbal instructions or verbalizations. Furthermore, when using a questionnaire, with concrete questions concerning the task, it seems useful to make the questions look like the task situation as much as possible. Questions that are too general or that seem unusual to the subjects will be perceived differently, with a different answering pattern. This will in turn lead to less valid protocols when used for knowledge extraction purposes.

REFERENCES

Berry, D. C., & Broadbent, D. E. (1984). On the relationship between task performance and associated verbalizable knowledge. Quarterly Journal of Experimental Psychology, 36 A, 209-231.

Boose, J. (1985). A knowledge acquisition program for expert systems based on personal construct psychology. International Journal of Man-Machine Studies, 23, 495-525.

Boose, J. (1988). A research framework for knowledge acquisition techniques and tools. In: J. Boose, B. Gaines, & M. Linster (Eds.), Proceedings of the Second European Knowledge Acquisition Workshop (EKAW'88). Sankt Augustin: Gesellshaft für Mathematik und Datenverarbeitung.

Broadbent, D. E., & Aston, B. (1978). Human control of a simulated economic system. Ergonomics, 21, 1035-1043.

Broadbent, D. E., Fitzgerald , P., & Broadbent, M. H. P. (1986). Implicit and explicit knowledge in the control of complex systems. British Journal of Psychology, 77, 33-50.

Burton, A. M., Shadbolt, N. R., Rugg, G., Hedgecock, A. P. (1988). Knowledge elicitation techniques in classification domains. In: Y. Kodratoff (Ed.), Proceedings of the 8th European Conference on Artificial Intelligence. London: Pitman Publishing.

Chi, M. T. H., Glaser, R., & Farr, M. J. (1988). The nature of expertise. Hillsdale, New Jersey: Lawrence Erlbaum Associates, Publishers.

Cooke, N. M., & McDonald, J. E. (1987). The application of psychological scaling techniques to knowledge elicitation for knowledge-based systems. International Journal of Man-Machine Studies, 26, 533-550.

Ericsson, K. A., & Simon, H. A. (1980). Verbal reports as data. Psychological Review, Vol. 87, No. 3, 215-251.

Ericsson, K. A., & Simon, H. A. (1984). Protocol analysis: Verbal reports as data. Cambridge, MA: MIT Press.

Haers, M. (1988). Knowledge extraction: The implicit - explicit distinction. I.W.O.N.L. Spilprogramma 'Expert Systems'. Interim Report September - December 1988. Psychological Report No. 94 (pp. 14-18). Leuven, University of Leuven, Laboratory of Experimental Psychology.

Hoffman, R. R. (1987). The problem of extracting the knowledge of experts from the perspective of experimental psychology. AI-Magazine, Vol. 8, No. 2, 53-67.

Newell, A., & Simon, H. A. (1972). Human problem solving. Englewood Cliffs, N.J.: Prentice-Hall.

Shaw, M. (1988). Problems of validation in a knowledge acquisition system using multiple experts. In: J. Boose, B. Gaines, & M. Linster (Eds.), Proceedings of the Second European Knowledge Acquisition Workshop (EKAW'88). Sankt Augustin: Gesellshaft für Mathematik und Datenverarbeitung.

Operation Presumption: Knowledge Acquisition by Induction

Hirokazu Taki and *Yuichi Fujii

ICOT Research Center, Institute for New Generation Computer Technology, 4-28, Mita 1-Chome, Minato-ku, Tokyo 108 Japan
Phone: +81-3-456-3192, Csnet:htaki%icot.jp@relay.cs.net

Abstract

This paper describes a knowledge acquisition support system which presumes knowledge fragments by induction. In this system, human problem solving knowledge is extracted as a set of operations. There are three knowledge acquisition phases in the knowledge acquisition process. The first is the expert model construction phase, in which problem solving knowledge is extracted as a set of operations and the type of each operation is defined. Operation types are prepared by this system. The second phase is the model instantiation phase, in which the system extracts detailed knowledge according to the operation types Sometimes, it is not easy to extract detailed information without knowledge seeds. Therefore, this system supports an inductive mechanism which makes operation presumptions in order to stimulate human experts to associate detailed knowledge. The third phase is the knowledge refinement phase, in which the system inference engine evaluates the knowledge base at the operation level to check whether the knowledge is sufficient. This paper introduces an interactive knowledge acquisition support system, EPSILON; knowledge representation, Expert Model; an interactive knowledge acquisition method, pre-post method; and an operation presumption method.

1. Introduction

There is a big problem in the building knowledge base of expert systems, called the knowledge acquisition bottle-neck. Knowledge acquisition support systems have been developed to solve this problem. There are three phases in the knowledge acquisition process; (1) the expert model modelling phase, (2) the expert model instantiation phase, and (3) the knowledge refinement phase. In the expert model modelling phase, a human problem solving model is extracted. In the expert model instantiation phase, knowledge, which is used in problem solving, is extracted. Knowledge consists of concepts, relations between concepts, and problem solving knowledge using concepts and their relations.

In the knowledge refinement phase, the knowledge acquisition system detects shortage and inconsistency of knowledge, and guides the human expert to modify the knowledge. Many knowledge acquisition support systems have assumed problem solving knowledge before knowledge acquisition. A problem solving strategy, which MORE [Kahn 85] assumes for expert systems, includes the usage of domain models for diagnosis tasks. A problem solving strategy, which ETS [Boose 84] assumes for expert systems, includes the classification method of items using knowledge in the form of a rating grid. EPSILON [Taki 87] has no problem solving strategy before knowledge acquisition. It extracts

---

* Current address is :
Engineering Strategy Planning Headquarters, Nippon Telegraph and Telephone Corporation, 1-6 Uchisaiwai-Cho, 1-Chome, Chiyoda-ku, Tokyo 100, Japan

problem solving knowledge first, then acquires other knowledge which is used by problem solving knowledge. These knowledge acquisition support systems use some methods to extract knowledge; ETS uses the personal construct theory to stimulate the human expert to associate constructs related to items. It also uses a refinement method which translates a rating grid into an implication graph and cluster trees to stimulate the human expert to find some shortage or inconsistency of knowledge. MORE uses a refine method which refines a domain model using eight interview strategies. EPSILON/One also supports a knowledge acquisition method, the pre-post method, to stimulate the human expert to remember operations of his expert jobs and to extract detailed knowledge of each operation using operation types.

However, it is difficult for us to prepare a general method to extract knowledge. To associate expert knowledge, we must prepare knowledge seeds. EPSILON/One asks the expert for detailed information of operations according to the operation type, but sometimes, the expert cannot answer or remember information. Humans can modify knowledge more easy than they can remember it. Therefore, to give EPSILON the capacity to extract operation information like this, we have been developing the operation presumption method, which supports making knowledge seeds to associate expert knowledge.

The following sections introduce knowledge representation for knowledge acquisition, Expert Model; a knowledge acquisition support system, EPSILON/One; and an operation presumption method using inductive inference.

2. Expert Model

This section explains the original idea of the Expert Model, and gives details of Expert Model structures.

2.1 Basic Idea of Expert Model

The Expert Model is based on two ideas: the simplified expert task model and analysis and grouping of diagnosis expert knowledge written in production rule form.

(1) Simplified Expert Task Model

Expert systems can be distinguished as different task types. According to "Building Expert Systems" [Hayes-Roth 83], there are 10 generic categories of knowledge engineering applications: interpretation, prediction, diagnosis, design, planning, monitoring, debugging, repair, instruction and control. We tried to define these categories as simple models, because simple expert task models provide the expert task images for the human expert to represent his knowledge. Two samples are introduced later.

Simple Diagnosis Task Model

Fig.1 shows a diagnosis task. Diagnosis tasks are represented in tree structures and their search algorithm in general AI architectures. However, the tree structure does not represent expert task images. A simpler model is a filter model. In Fig.1, G1 denotes the possible results of trouble positions, F1 is a filter to select real results, and G2 denotes the results of a diagnosis task. Fig.2 also shows a diagnosis task, but this model has some sub-

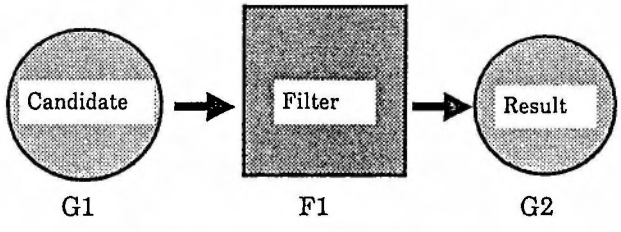


Fig.1 Simple Diagnosis Task Model

tasks which are the same as these of Fig.1.

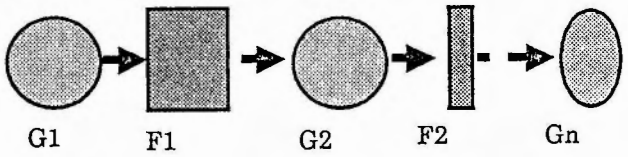


Fig.2 Simple Diagnosis Task Model(No.2)

Simple Design Task Model

Fig.3 shows a very simple design task. Design tasks are very complex; therefore, this model is actually a sub-task of a design task. It consists of a part modification phase, a part assembly phase and a good selection result phase. G4 and G5 are design-part groups, F4 is a modifier, G10 is a group of modified G4 parts, F5 is an assembly or combination operation, G11 is an interim result group, F6 is a selector, and G12 is a group of design task results. (G5 should not be modified.) We have considered simplified expert task models and created a model (operation) which is constructed with a source element-group, evaluators (e.g. select filter) and a destination element-group.

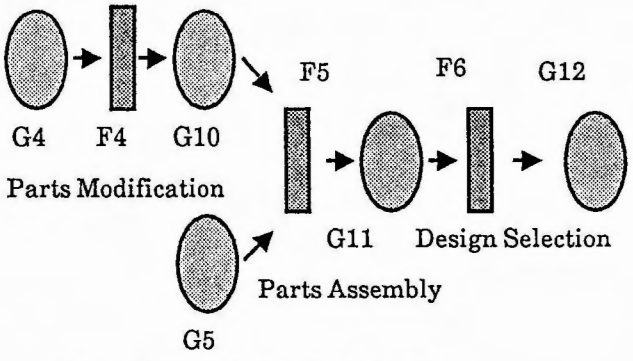


Fig.3 Simple Design Task Model

(2) Study from Production Rules of Diagnosis Expert Systems

A production rule is a general knowledge representation for expert systems. Therefore, knowledge engineers must have techniques of rule writing to represent many kinds of knowledge. We assumed that those techniques may appear in the form of rules. Seven operations have been discovered in production rule sets: SELECTION, CLASSIFICATION, ORDERING, COMBINATION, TRANSLATION, INPUT and OUTPUT. The combined result of both ideas is the generic operation.

2.2 Operation Types

The Expert Model is a specialized knowledge representation to collect expert task smoothly from human experts. It is not necessary for the knowledge acquisition process to provide a general knowledge representation, i.e. , production rule form. This section explains the components of the Expert Model.

2. 2. 1 Generic Operations

Generic operations are kernel knowledge representations in the Expert Model. The generic operations have been derived from simple expert task models and by analyzing production rules of diagnosis expert systems. The benefits of these frameworks are explained as knowledge acquisition methodology in section 3.

There are seven generic operations: SELECTION, CLASSIFICATION, ORDERING,

COMBINATION, TRANSLATION, INPUT and OUTPUT. Each operation has one or more source groups, destination groups and evaluator groups.

SELECTION Operation

The SELECTION operation picks up elements from the source group according to the selection conditions of the evaluators. Fig.4 shows an example of the SELECTION operation.

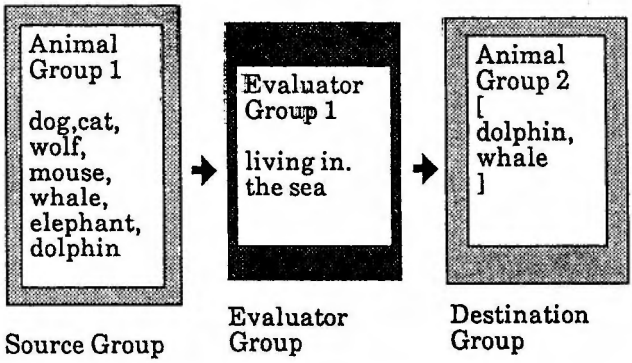


Fig.4 Example of SELECTION operation

CLASSIFICATION Operation

The CLASSIFICATION operation separates source elements into groups. The upper SELECTION operation is an example which is a special case of CLASSIFICATION. Fig.5 shows an example of the CLASSIFICATION operation.

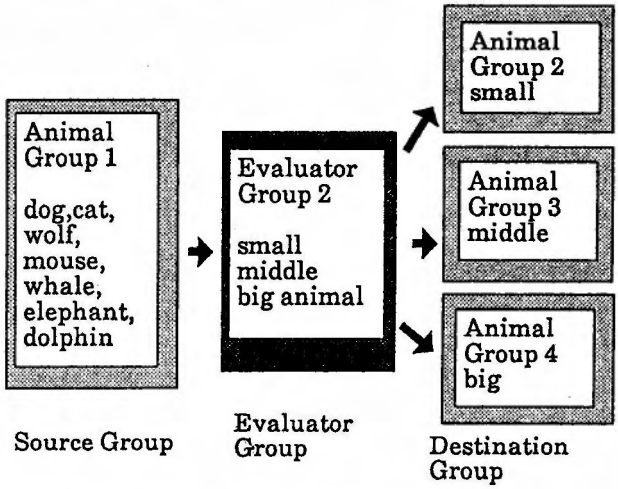


Fig.5.Example of CLASSIFICATION operation

ORDERING Operation

Elements of groups take turns to evaluate. The ORDERING operation sors elements of the source group and makes a destination group, in which elements take their turns in order of the ordering condition.

COMBINATION Operation

The COMBINATION operation assembles source elements into new elements. Generally, there are multiple sources, as shown in Fig.6.

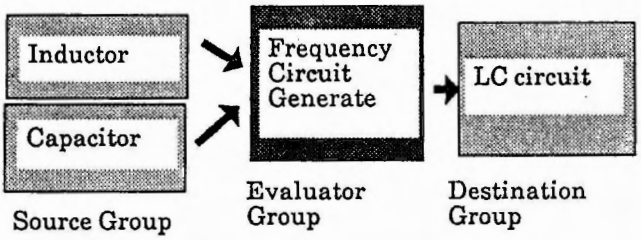


Fig.6.Example of COMBINATION operation

TRANSLATION Operation

The TRANSLATION operation is used for data abstraction and data interpretation. In this operation, elements of the source group are translated and the destination group is generated. New elements must be generated, or attributes of elements must be added or changed.

INPUT Operation and OUTPUT Operation

These are interaction operations between an expert system and a user (or a measuring device). The INPUT operation makes a new group without source groups. The OUTPUT operation does not make a destination group. It gives source group information to the user according to the output process.

2.2.2 Elements

Elements consist of items and their attributes. Elements are frame-based knowledge representations which have attribute-slots and an inheritance, but they do not have the attached procedures as demons. Each element must belong to one or more element groups. An element is accessed by one or more generic operations.

2.2.3 Evaluators

Evaluators consist of conditions and actions, and may include the procedural process. Sometimes, the evaluator group is a source group of a generic operation. For example, in a constraint relaxation problem of a planning task, a big evaluator group which has many conditions as constraints must be modified to a small evaluator group which has fewer conditions. This small evaluator group is used to solve the planning task.

2.2.4 Meta-Operation-Control: Meta-Script

The expert tasks have their scenario for solving problems, therefore, this meta-operation control is called Meta-Script in the Expert Model. It is a representation to describe meta-level knowledge. Fig.7 shows its location in the knowledge base.

The Expert Model is recognized as an object-oriented model, and a generic operation is an object. The communication between a generic operation and other operations is a message passing process, which has two types in the Expert Model: the inheritance and the post operation call.

3. Knowledge Acquisition Support System : EPSILON

3.1 EPSILON System Over View

The knowledge acquisition support system, EPSILON consists of three sub-systems. They are a knowledge elicitation system, an expert model inference engine and a knowledge refinement system.

The knowledge elicitation system interviews the expert directly, and extract knowledge in form of Expert Model. This sub-system uses pre-post method to acquire knowledge by interview.

The knowledge refinement system checks lacks and inconsistency of knowledge base.

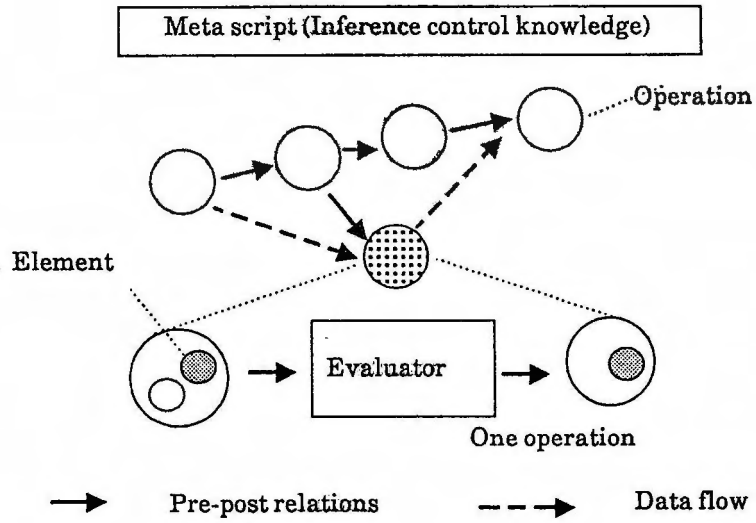


Fig.7 Hierarchy of the Expert Model

Extracted Expert Model is translated into logical knowledge representation and checked its inconsistency. Lacks of it is checked to compare Expert Model and other knowledge. For exmaple, if the Expert Model is knowledge for diagnosis of a machine, it is compared to the machine design information which represents its parts and the structure.

The expert model inference engine evaluates Expert Model. The expert can check his knowledge to observe behaviors of operations. EPSILON system structure is shown in Fig. 8.

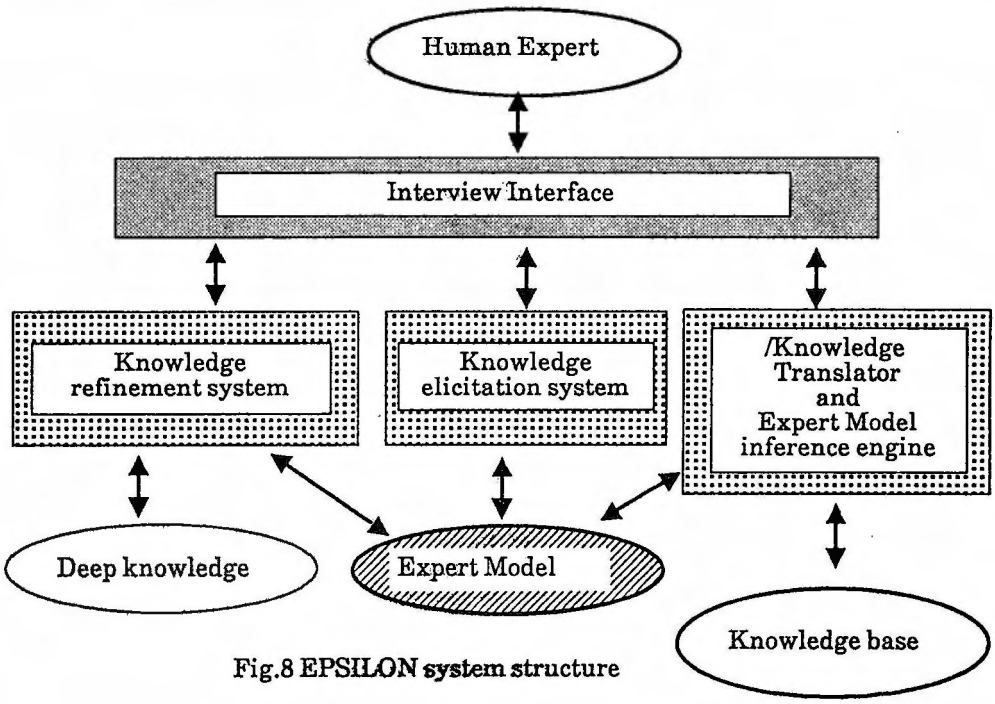


Fig.8 EPSILON system structure

3.2 Knowledge Acquisition Method : Pre-Post Method

The pre-post method is a problem solving model construction and model instantiation method. This method has a few model building strategies. The main strategy of this method stimulates a human expert to remember pre-post operations which are associated with a focal operation. The human expert can easily answer what operations are necessary before or after the operation. For example, when the car does not move, the human is asked *"What do you do before checking the engine ?"*. He can easily answer, *"I have to check the remaining gas" or "I have to check the battery"*. The following steps are knowledge acquisition steps based on the pre-post method.

Step 1: Collecting several human expert operations at random. It is not necessary for each operation to have any relation to the others. These operations are used as the starting points of knowledge acquisition.

Step 2: Making questions to obtain pre and post operations for each collected operation. This step continues until no new operations are extracted from the human expert. The same process is repeated for new operations.

Step 3: Many operations have been gathered and they have pre/post information which operations have to execute before or after operations. This step is the pre/post relation check step. One way of checking is to display pre/post relations graphically for the human expert.

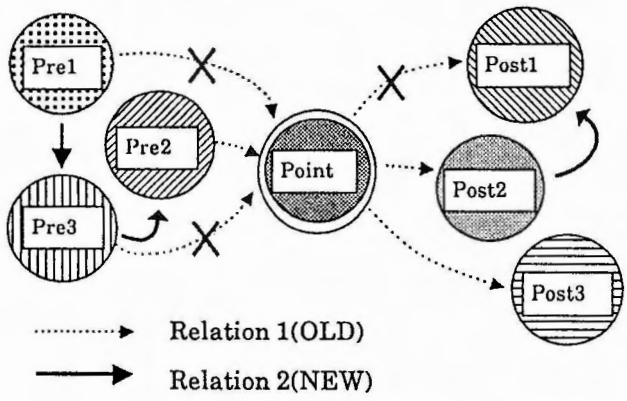


Fig.9 Pre-post relation example

Fig.9 shows pre-post relations. Relation1 shows only that Pre1 to Pre3 are pre-operations of a point operation and Post1 to Post3 are its post-operations. In relation2, there are new orders among pre-post operations defined by the expert. Pre1 is a pre-operation of Pre3. Pre2 is a post-operation of Pre3. Post1 is a post-operation of Post2. The more pre-post information collected, the clearer pre-post relations.

The sequence of some operations may be unclear. We treat the group of these operations as a weak sequence strategy group. When an operation of this group is finished, sometimes the result of the operation affects the other operations of this group. It is important for this group to obtain constraint information.

Step 4: This step is the instantiation step of each operation. The human expert must select a generic operation type to define the role of each operation. Sometimes, operations defined by the human expert are big tasks which contain several sub-operations. To distribute these big tasks, it is important to ask *"What sub-operations exist in XX (an operation name)? Please answer at random"*. After this question, pre-post operations must be extracted for sub-operations. Then, the Step 3 process is executed again.

Note: From step 1 to step 4, a problem solving model is made. From step 5, the details of operations in this model is extracted.

Step 5: This step merges operations. According to pre-post information and generic operation types, some operations are merged to one generic operation.

Step 6: Each operation has evaluators. This step defines these evaluators. It is not difficult to do this process, because the role of the operation is clear and its function has been identified by its generic operation type.

Step 7: The elements of the source group are defined in this step. Sometimes, it is difficult to decide the elements and group of a focal operation, because they are decided dynamically after its pre-operation has been executed. Therefore, the human expert must search for the candidates of the source group from the result (destination group) of its pre-operations.

Step 8: Attributes and attribute values of the elements are defined in this step. Each element must have the attribute evaluated by the evaluators. Generally, elements flow through a lot of operations, and need the attribute which is evaluated by the evaluators in the flow.

Other Steps: There are other steps in pre-post method. They are operation blocks making step to group operations with same inference control, and operation inference control definition step.

4. Operation Presumption

This operation presumption method makes the details of operation from input data, input element examples, output data, and output element examples of an operation inductively. This section explains the details of operation presumption for four operation types (selection, classification, translation and ordering). It is not necessary to make knowledge seeds to associate expert knowledge for the other operation types (input, output and combination).

4.1 Operation Presumption by Inductive Inference

Induction is an inference which makes more general rules or concepts from examples. It decides the generalization level of knowledge to limit a version space [Mitchell 78] from positive and negative examples. For example, in a selection operation, an input element group is a set of all examples. An output element group is a positive example set of this operation. Elements which are not selected by this operation are negative examples. An operation evaluator can be presumed from input and output elements. However, it is difficult to make a good generalization level of the operation evaluator from a few examples [Valiant 84]. Therefore, this system uses the results of induction for knowledge seeds. If the expert does not remember the details of an operation easily, this is a useful knowledge acquisition method.

4.2 Selection and Classification Criteria Presumption

To presume selection or classification criteria, attributes and criteria formulae (logical or mathematical formulae) must be decided. If attributes to be focused on are multiple, logical relations between criteria of these attributes must be decided. To explain the presumption examples, element groups (EGi: i-th element group) and elements (ELj: j-th element) are defined as follows.

An element group is shown as a set of elements.

$$EGi = \{ELj, ELj+1, ..., ELk\}$$

An element is shown as a list of pairs of an attribute (ATn: n-th attribute) and an attribute value (VAn: n-th attribute value).

$ELj = (ATn = VAn) \wedge (ATn+1 = VAn+1) \wedge \ldots$.

Then, a selection criterion includes pairs of an attribute and its value in positive examples, but does not include pairs of an attribute and its value in negative examples. We assume that a human expert uses simple and general criteria in preference to complex and special ones when making a selection. For this reason, the system begins with simple criterion generation. Candidates for selection criteria are generated as follows.

Step 1: Picking up attributes in all elements.

Step 2: Making two attribute value sets for each attribute in positive examples and negative examples. If an element has an attribute, the system regards the element which has a pair of the attribute and an attribute value as being "undefined". A value set made from positive examples is called a positive attribute value set (PVS) for an attribute. A value set made from negative examples is called a negative attribute value set (NVS) for an attribute.

Example 1: Positive examples:

$EL1 = (AT1 = VA1) \wedge (AT2 = VA2)$

$EL2 = (AT1 = VA1) \wedge (AT3 = VA3)$

Negative example:

$EL3 = (AT2 = VA4) \wedge (AT3 = VA3)$

PVS for AT1 = (VA1), NVS for AT1 = (undefined)
PVS for AT2 = (VA2 $\vee$ undefined), NVS for AT2 = (VA4)
PVS for AT3 = (undefined $\vee$ VA3), NVS for AT3 = (VA3)

Step 3: Searching attributes for which an intersection of PVS and NVS is an empty set. If there are attributes which satisfay this condtion, the system makes some candidates for selection criteria and shows them to the human expert as **Step 6.** In example 1, attributes AT1 and AT2 satisfy this condition, but attribute AT3 dose not. If there are no attributes which satisfy this condition, the system processes **Step 4.**

Step 4: Making two attribute value sets for multiple attributes in positive and negative examples. Elements of these sets are conjunctions which consist of a pair of an attribute and its value. A set made from positive examples is called a multiple positive attribute values set (MPVS). A set made from negative examples is called a multiple negative attribute values set (MNVS). Combinations of attributes begin with attributes which have the lowest number of "undefined" in PVS and NVS.

[Example 2] Positive example:

$EL4 = (AT5 = VA5) \wedge (AT7 = VA7)$

$EL5 = (AT5 = VA5) \wedge (AT6 = VA6) \wedge (AT8 = VA8)$

Negative example:

$EL6 = (AT5 = VA5) \wedge (AT6 = VA9)$

$EL7 = (AT6 = VA6) \wedge (AT7 = VA10)$

PVS for AT5 = (VA5), NVS for AT5 = (VA5 $\vee$ undefined)
PVS for AT6 = (undefined $\vee$ VA6), NVS for AT6 = (VA9 $\vee$ VA6)
PVS for AT7 = (VA7 $\vee$ undefined), NVS for AT7 = (undefined $\vee$ VA10)
PVS for AT8 = (undefined $\vee$ VA8), NVS for AT8 = (undefined)
AT5 and AT6 have one undefined values.
AT7 and AT8 have two undefined values.

Order of Combinations: AT5 ∧ AT6, AT5 ∧ AT7, AT6 ∧ AT7, AT5 ∧ AT8, AT6 ∧ AT8, AT8 ∧ AT7, AT5 ∧ AT6 ∧ AT7, ...

MPVS for AT5 and AT6 = (AT5 = VA5)∧(AT6 = undefined))∨((AT5 = VA5)∧(AT6 = VA6)

MNVS for AT5 and AT6 = (AT5 = VA5)∧(AT6 = VA9))∨((AT5 = undefined)∧(AT6 = VA6)

Step 5: Checking whether an intersection of an MPVS and an MNVS is an empty set. If the MPVS and MNVS pair satisfies the condition, the system makes some candidates for selection criteria and show them to the human expert as **Step 7**. If the pair dosen't satisfy the condition, the system checks another pair.

Step 6: Generating three candidates from PVS and NVS for selection criteria. They are an expression of an attribute and its PVS, a negative expression of an attribute and its NVS, and a conjunction of these two expressions.

In example 1, they are shown as follows.

[Attribute AT1] Type 1: AT1 = VA1
Type 2: AT1 ≠ undefined
Type 3: (AT1 = VA1)∧(AT1 ≠ undefined)

[Attribute AT2] Type 1: AT2 = VA2 ∨ undefined
Type 2: AT2 ≠ VA4
Type 3: (AT2 = VA2 ∨ undefined)∧(AT2 ≠ VA4)

The system shows type 1 expression first. If the human expert requires another expression, the system shows type 2 expression second and type 3 expression third. If the human expert finds new selection criteria because of these hints, the expert modifies them directly. If the expert rejects the expression of this form expression, the system return to **Step 4** to make other candidates.

Step 7: Generating three candidates from MPVS and MNVS for selection criteria. They are an expression of an attribute and its MPVS, a negative expression of an attribute and its MNVS, and a conjunction of these two expressions.

In example 2, they are shown as follows.

[Attribute AT5 and AT6]
Type 1: ((AT5 = VA5)∧(AT6 = undefined))∨((AT5 = VA5)∧(AT6 = VA6))
That is: (AT5 = VA5)∧(AT6 = undefined ∨ VA6)
Type 2: ((AT5 ≠ VA5)∨(AT6 ≠ VA9))∧((AT5 ≠ undefined)∨(AT6 ≠ VA6))
Type 3: (AT5 = VA5)∧(AT6 = undefined ∨ VA6)∧
((AT5 ≠ VA5)∨(AT6 ≠ VA9))∧((AT5 ≠ undefined)∨(AT6 ≠ VA6))

The system shows these candidates as in **Step 6**. If the human expert rejects the expression of this form, the system return to **Step 4** to make other candidates.

This method is based on logical relation. So the current version of the selection operation presumption algorithm doesn't make an expression "AT1 >= 10" from an original expression "AT1 = 10, 20, 100, ...".

Example 3: *Animal selection operation Input and output element groups are defined as follows.*

Input EG = {lion, whale, penguin, salmon, dolphin, elephant}
Output EG = {whale, dolphin, salmon}

Elements and their contents are defined as follows.

lion = (habitat = plain)∧(legs = 4)∧(genus = mammalia)
whale = (habitat = sea)∧(has = fins)∧(genus = mammalia)
penguin = (habitat = seaside)∧(legs = 2)∧(has = wings)∧(genus = birds)
salmon = (habitat = sea)∧(has = fins)∧(genus = fish)
dolphin = (habitat = sea)∧(has = fins)∧(genus = mammalia)
elephant = (habitat = plain)∧(legs = 4)∧(genus = mammalia)

A normal logical selection criterion is derived from these examples.
It is :

(((habitat = sea) ∧ (has = fins) ∧ (genus = mammalia)) ∨
((habitat = sea) ∧ (has = fins) ∧ (genus = fishes)) ∨
((habitat = sea) ∧ (has = fins) ∧ (genus = mammalia))) ∧
¬(((habitat = plain) ∧ (legs = 4) ∧ (genus = mammalia)) ∧
((habitat = seaside) ∧ (legs = 2) ∧ (has = wings)) ∧ (genus = birds)) ∧
((habitat = plain) ∧ (legs = 4) ∧ (genus = mammalia))

An arranged criterion is :

(habitat = sea) ∧ (has = fins)∧ (genus = mammalia ∨ fish)∧
((habitat ≠ plain)∨(legs ≠ 4)∨(genus ≠ mammalia))∧
((habitat ≠ seaside)∨(legs ≠ 2)∨(has ≠ wings)∨(genus ≠ birds))

This logical formula is a candidate knowledge seed of this operation. However, the expert focuses on one or a few attributes to select or classify elements. This formula is most specialized criterion. So the system tries to propose more general ones.
In example 3, PVS and NVS are shown as follows.

PVS for habitat = (sea), NVS for habitat = (plain ∨ seaside)
PVS for legs = (undefined), NVS for legs = (2 ∨ 4)
PVS for has = (fins), NVS for has = (undefined ∨ wings)
PVS for genus = (mammalia ∨ fish)
NVS for genus = (mammalia ∨ birds)

Pairs of PVS and NVS for attributes "habitat", "legs" and "has" have no intersections. The pair of PVS and NVS for "habitat" has no undefined value, so the system makes following candidates.

(1) habitat = sea
(2) habitat ≠ plain and seaside
(3) (habitat = sea)∧(habitat ≠ plain ∧ seaside)

If the expert didn't use these criteria, the system generates the following candidates according to our method.

(4) legs = undefined
(5) legs ≠ 2 and 4
(6) (legs = undefined) ∧ (legs ≠ 2 ∧ 4)
:
(i) (habitat = sea)∧(legs = undefined)
(i+1) ((habitat ≠ plain) ∨ (legs ≠ 4))∧((habitat ≠ seaside)∨
(legs ≠ 2))
(i+2) ((habitat = sea)∧(legs = undefined))∧
((habitat ≠ plain) ∨ (legs ≠ 4))∧((habitat ≠ seaside)∨
(legs ≠ 2))
:

4.3 Translation Equation Presumption

A translation operation makes a new pair of an attribute and its value to refer to one or

more attribute values. In the following examples, referred attributes are assumed in the same elements. There are two kinds of translation operations: translating one symbol into another, and translating one number into another using numerical calculation.

Example 4: *Make a new attribute and its value*

(Make an area of a box)

Input EG = {box}

Output EG = {box}

box in input EG = (height = 20)∧(length = 5)

box in output EG = (height = 20)∧(length = 5)∧(area = 100)

If a human is given example 4, he compares the input element group and the output element group, and quickly makes "area = height * length", because he has knowledge about area. If he is given an example that makes (c = 100) from (a = 20) and (b = 5), he also makes "c = a * b" easily. There are many candidates which satisfy this relation.

area = height * length
area = height * 5
area = height * 6 - length * 4
area = length * 20

If the element "box" has (weight = 100), the human makes "area = weight". There are too many induction results to narrow down. Before knowledge acquisition, it is difficult for knowledge acquisition systems to know the relations between attributes (for example, relations among area, length and height). A set of attributes in an input element group related to the generated new attribute is first determined for presumption. The intersection of attributes of input elements must be shown for the expert. The expert selects focused attributes from this set. However, sometimes, the expert checks different attributes for each input element. Then, other attributes are also shown. In this example, after the expert selects attributes "height" and "length", he forms relations between these attributes. After attributes are selected, there are many translation formula candidates.

If the expert can define the form of a translation formula, this presumption is a problem which decides the formula parameters. If enough examples are given, this system solves simultaneous equations. The problem of deciding on a translation formula is difficult. Therefore, the current version of the presumption algorithm supports only the selection of possible candidates of attributes by forming on intersection of attribute sets of input elements when multiple examples are given.

Next, we discuss the problem of presuming symbol exchange. Focused attributes are also important in this case. In output elements, the system must focus on new attributes or attributes which have new values in output elements. There is a strong possibility that an attribute which has a new value is a candidate. This heuristic is useful for selecting attributes in this operation type.

Example 5(a): *Change attribute value*

Input EG = {box 1, box 2, box 3}

Output EG = {box 1, box 2, box 3}

box 1 in input EG = (area = 50)∧(place = on-table)
box 2 in input EG = (area = 60)∧(place = on-floor)
box 3 in input EG = (area = 100)∧(place = in-garden)

box 1 in output EG = (area = small)∧(place = on-table)
box 2 in output EG = (area = small)∧(place = on-floor)
box 3 in output EG = (area = large)∧(place = in-garden)

Example 5(a) shows a translation operation focusing on the attribute "area". The following logical formulae are generated easily in example 5.

(area = 50)$\vee$(area = 60) --> (area = small)
(area = 100) --> (area = large)

These formulae handle numbers for translation criteria. Using a heuristic in which one or more numbers select the translation criterion, these formulae can be generalized. The comparison result is :

(area = 50) < (area = 60) < (area = 100)

(area = 50) and (area = 60) are translated into the same value (area = small), which means that the necessary criterion is between 60 and 100. Then, EPSILON proposes the following candidates.

(area <= 80) --> (area = small)
(area > 80) --> (area = large)

The expert checks these translation formulae forms the following, for example.

(area <= 90) --> (area = small)
(area > 90) --> (area = large)

Example 5(b): *Change attribute value*
Input EG = {box 1, box 2, box 3}
Output EG = {box 1, box 2, box 3}

box 1 in input EG = (area = 50)$\wedge$(place = on-table)
box 2 in input EG = (area = 60)$\wedge$(place = on-floor)
box 3 in input EG = (area = 100)$\wedge$(place = in-garden)

box 1 in output EG = (area = 50)$\wedge$(place = in-house)
box 2 in output EG = (area = 60)$\wedge$(place = in-house)
box 3 in output EG = (area = 100)$\wedge$(place = out-of-house)

Example 5(b) handles symbol translation focusing on the attribute "place". The following logical translation formulae are generated, but cannot be generalized.

(place = on-table) or (place = on-floor) --> (place = in-house)
(place = in-garden) --> (place = out-of-house)

4.4 Ordering Criteria Presumption

There is a process order, which represents a list of elements, in each element group. An inference engine of EPSILON has two modes for problem solving: all solution search and partial solution search. The inference engine uses this order for partial solution search. The ordering operation sorts elements according to the mathematical or symbol order, for example, [short, middle, long]. The mathematical order may be ascending or descending.

Example 6: *Sorting using symbolic ordering*
Input EG = {box 1, box 2, box 3}
Output EG = {box 2, box 3, box 1}

box 1 = (size = small)
box 2 = (size = middle)
box 3 = (size = large)

The order of symbols of example 4 is the order list of values of an attribute (size).
Obtained criteria: size = [small, middle, large]

Example 7: Sorting using numerical ordering
Input EG = {box 1, box 2, box 3}
Output EG = {box 2, box 3, box 1}

box1 = (size = 150)
box2 = (size = 120)
box3 = (size = 130)

The order of example 7 is made from numerical comparison of an attribute (size).
Obtained criteria: Ascending order for size

If focused attributes are multiple, then a translation operation must be defined before this ordering operation to translate values of multiple attributes into one value of one attribute.

4.5 Knowledge Acquisition using Operation Presumption

There are many learning algorithms, but most of them are for special knowledge. It is therefore difficult to extract all knowledge of expert systems using only the learning algorithms which have been developed to date. However, these methods are useful for making knowledge seeds of interactive knowledge acquisition systems. It is important for us to apply learning methods to knowledge acquisition. In our method, if the induction result class is too wide (general), we must search for presumption heuristics. It is also important to ask good questions to limit the class of the criteria [Angluin 86]. Induction heristics are related to non-monotonic reasoning [Nicolas 89].

5. Summary

This paper introduced the knowledge acquisition support system, EPSILON, and a presumption method to make knowledge seeds. We proposed Expert Model as knowledge representation for knowledge acquisition, used in EPSILON. Expert Model consists of operations, so it is extracted in operation form. EPSILON supports an interview method and a presumption method to extract operation details. This operation presumption is a syntactical function, but we must support heuristics for effective presumptions. The currently implemented EPSILON is EPSILON/One, which supports the non-hierarchical Expert Model and pre-post method. EPSILON/One is implemented on a sequential inference machine (PSI-2), developed by ICOT, in an object oriented logic programming language ESP[Chikayama 84].

6. Acknowledgment

We wish to express our thanks to the member of 5th Laboratory at ICOT, Mr. Kazuhiro Tsubaki and Mr. Eiichi Horiuchi, the chief of 5th Laboratory, Mr. Kenji Ikoma, the members of the Knowledge base System Architecture - Knowledge Acquisition and Design - Sub Working Group, its head, Professor Riichiro Mizoguchi and Professor Norihiro Abe. We would like to thank the members of the Learning and Non-monotonic Reasoning System Research Group (LANRS) at ICOT and Dr. Koichi Furukawa. Finally, we must express our thanks to Dr. Kazuhiro Fuchi, Director of ICOT Research Center, who provided us with the opportunity to conduct this research in the Fifth Generation Computer Systems Project.

References

[Angluin 86] Angluin, D. :Types of queries for Concept Learning, TALEU/DCS/TR-479, June 1986

[Bennett 85] Bennett, J. : ROGET: A Knowledge-Based System for Acquiring the Conceptual Structure of a Diagnostic Expert System, Journal of Automated Reasoning 1, 49-74, 1985

[Boose 84] Boose, J. : Personal Construct Theory and the Transfer of Human Expertise, in Proceedings of the National Conference on Artificial Intelligence, Austin, Texas, 1984

[Chandrasekaran 86] Chandrasekaran, B. : Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design, IEEE Expert, Fall 1986

[Chikayama 84] Chikayama, T. : Unique Features of ESP, International Conference on Fifth Generation Computer Systems, November 1984

[Clancey 85] Clancey, W. : Heuristic Classification, Artificial Intelligence 27, 1985

[Genesereth 86] Genesereth, M. , Nilsson, N.: Logical Foundations of Artificial Intelligence Los Altos, CA: Morgan-Kaufmann, 1986

[Hays-Roth 83] Hays-Roth, F. , Waterman, D. A. , Lenat,D. B. : Building Expert Systems, Addison-Wesley, 1983

[Iwashita 86] Iwashita, Y. , et al. : Knowledge Acquisition and Learning in Case Studies of Expert System Development, ICOT TR-0204, 1986

[Kahn 85] Kahn, G. , Nowlan, S. and McDermott, J. : Strategies for Knowledge Acquisition, IEEE transactions on Pattern Analysis and Machine Intelligence 7(5), 1985

[Nicolas 89] Nicolas, Helft, Induction as Nonmonotonic Inference, KR89,1989

[Miki 87] Miki, M. and Taki, H. : The Knowledge Acquisition System with Fault Tree and Diagnostic Flow Interface, Proceedings of International Workshop on Industrial Applications of Machine Vision and Machine Intelligence, Tokyo, 1987

[Mitchell 78] Mitchell, T.M. : Version Spaces: An Approach to Concept Learning, Stanford Technical Report STAN-CS-78-711, HPP-79-2

[Taki 87] Taki, H., Tsubaki, K. and Iwashita, Y. : EXPERT MODEL for Knowledge Acquisition, the Proceedings of IEEE Expert Systems in Goverment Conference, 1987

[Taki 88] Taki, H. : Knowledge Acquisition by Observation, Proceedings of Ineternatinal conference of Fifth Generation Computer Systems, 1988

[Valiant 84] Valiant, L.G. : A Theory of the Learnable, CACM, Vol. 27, No. 11, pp1134-1142, 1984

THE SCIENCE OF KNOWLEDGE:
A Research Programme for Knowledge Engineering

T. R. Addis

Knowledge Systems Group
Computer Science Department
University of Reading

Abstract

Artificial Intelligence and Knowledge Engineering together have generally been pursued as an engineering subject. Attempts at reformulating this engineering activity into a scientific paradigm have not been successful. In particular, these reformulations have been unable to explain: the difficult problem of feature acknowledgement, the logical inconsistency in using deduction for ampliative reasoning and the character of knowledge-based systems. These reformulations have even failed to define Intelligence. A "priming" theory is proposed that makes clear some of the terms used in Knowledge Engineering and provides a framework in which knowledge and its representations can be discussed. The theory is intended to start a research programme that will develop along traditional scientific lines.

1. THEORIES, MODELS AND ARTIFACTS

Many AI workers consider that a program can be viewed as a theory (eg Langley 1988). It has been suggested that experiments on a program are equivalent to the scientific method where experiments are created in order to validate or disprove a theory (ie to determine its truth). This view seems particularly relevant when considering knowledge acquisition with the purpose of simulating the performance of an expert. In this case, the program simulates the expert and hence predicts the actions of the expert under given conditions; it is this capacity to predict that makes the program appear as a theory (Addis, 1987).

An engineering simile can be examined in order to draw the fine distinction between a program and a theory. A program can be considered to be an artifact such as a bridge. Then to propose that a program is a theory would be equivalent to saying that a bridge built over a river is a theory and the bridge can be tested to determine its validity (its truth?). There is no doubt that it is of great utility and good engineering practice to carry out tests over the range of purposes for which the bridge has been constructed; tests that record the behaviour of the bridge under different conditions. Testing is of great utility since if ever a bridge of that kind were to be constructed again then its characteristics would be well known. Testing is good engineering practice since predicted behaviour must be verified and margins of safety confirmed.

If tests were to be carried out for every different kind of artifact then a large collection of useful facts and common features would emerge. But what could be meant by a true or valid artifact and what would dispose of such an artifact? From where does the predicted behaviour come? A bridge may not meet its specifications and it may collapse under a certain load or fail to span the gap. But specifications are arbitrary and lighter loads can be recommended or the design used for narrower spans. A specification is decided by a designer or client upon issues many of which are external to the finished product.

It is also good engineering practice to carry out tests on models as well as completed structures. In this case a model acts like a theory of the artifact in the sense that it will predict behaviour. A theory, on the other hand, evolves to explain phenomena, to propose new observations (and hence tests) as well as to make predictions of behaviour.

In the case of an expert system or a knowledge-based program the program is created not just to perform a task but to simulate an expert (or a consensus of experts); the expert is *modelled.* The program can be tested and shown to be correct or incorrect (true or false) since validation comes from a direct comparison of program performance against expert performance. The construction of knowledge structures and the performance of the program will often suggest new observations to be made or questions to be asked. However, as a theory it is confined to a very narrow domain; and even within this domain its explanatory potential is limited.

Newtonian mechanics is an example of a *scientific* theory in that it has wide applicability not only to artifacts such as bridges but also to the motion of planets, the behaviour of gases and the mechanical structure of animals. Consequently, novel designs of bridges can be assessed and their behaviour predicted long before they are built. However, such is the complexity of the world that frequently only simple approximations of these structures are ever calculated in practice and then tests are performed on models; models that are constructed in the light of the theory. Thus, a model acts as a means of translating a specific manifestation of a theory into a testable artifact.

The important difference between a program (when considered as a model) and a theory is that a program is restricted to a particular realization of a concept whereas a theory has much wider applicability; a theory unifies. A theory used in engineering predicts the behaviour of the artifact, enables understandable comparisons to be made between different constructs and exposes the important observations to be made. Although a program can be viewed as a theory it is in fact a model and a model requires a theoretical framework in which to be constructed. Such a framework is a theory of knowledge.

2. THEORY DRIVEN KNOWLEDGE ENGINEERING

A theory of knowledge is needed to fulfil the following requirements and purposes:

1 **Completeness** - One of the essential elements of a theory is a taxonomy. In Newtonian mechanics there are concepts such as force, momentum, velocity, energy and acceleration. These concepts form a taxonomy by introducing into the world distinctions that can be measured and used to support the theory. An equivalent taxonomy for knowledge will ensure that any system that is designed will have the capacity to represent all the necessary knowledge that will be appropriate for its intended function.

   Some distinctions are primary (primitives) and their effects are observed indirectly (eg force, mass). Other distinctions are complex and are defined in terms of primary distinctions through relationships (eg acceleration, weight).

2 **Knowledge Elicitation** - The taxonomy will specify all the different classes of knowledge where each class will suggest a means of elicitation.

3 **System Design** - A theory will specify the structure of a system in order for it to achieve a specific function.

4 **Coordination of Development** - Any theory, even if it is obviously weak or wrong, will provide the basis for research and development. A theory will suggest experiments and observations; it will act as a stimulus for further research. A poor theory poses a challenge to others to create a better one and it introduces a common language through which scientists may communicate; it provides a means of comparison and growth.

Workers in the field of knowledge engineering have already adopted a theoretical framework by adapting personal construct theory (Kelly 1955). Methodologies have been explored that reflect psychological or cybernetic concepts (Johnson 1987) but none of these seems to develop into a vigorous scientific programme.

The need for a theory is also reflected in machine learning (ML) in the fact that many designers of systems are currently referring to theories drawn from different stances of the philosophy of knowledge. The quest for clearer goals in the definition of reasoning and learning systems has brought many researchers to review the philosophical works on reasoning and science. For example, Langley et al (Langley 1987) have made extensive use of the work on induction of

such philosophers as John Stuart Mill, Karl Popper, Thomas Kuhn and Imre Lakatos; and the Kuhnian paradigm shifts have been used for the justification of non-cumulative learning in explanation-based learning (Emde 1987).

Elsewhere the work of philosophers and logicians has been used solely to find terms with which to describe the types of inference occurring within AI programs. For example, Charniak and McDermott introduce their versions of abduction and induction (Charniak, 1985). Further definitions of abduction and induction are provided by Johnson and Keravnou for their analysis of inference in Expert Systems (Johnson, 1985). These definitions are based upon the philosophy of Peirce (Peirce, 1958). They describe abduction as the creation of new explanatory hypotheses and induction as the process of support or refutation of abductively inferred hypotheses.

However, despite this reference to established philosophical work the state of the art is confusing. There is little or no consensus on the nature of inference in machine reasoning and machine learning. The philosophical views have been constructed with different objectives to those of AI researchers. It is thus not surprising that the definitions of knowledge and reason are not directly applicable without some essential interpretation. What is needed is the development of a theory and the evolution of a philosophy of technology that is concerned with the relationship between knowledge, man and machine.

3. PUZZLING PHENOMENA

Research only makes sense in the context of a purpose. It is a long held and mistaken belief that knowledge can be pursued for knowledge's sake. In the case of many scientific endeavours such as subatomic physics or astronomy, the purpose is to achieve an explanation of the world that is elegant. An explanation is not any old tale (otherwise a fable would be considered a scientific theory) but one that enables sound predictions of observable events.

Such explanations (theories, hypotheses) can be used for the stimulation of development. However, what is observable depends upon the explanation (the theory). Large and expensive pieces of equipment such as cyclotrons and telescopes are constructed **in order** to observe the otherwise unobservable. These extensions of the human senses will only stop when an elegant theory fits all the measurements suggested by the theory. Physics nearly achieved this ideal at the beginning of the century until a certain Michelson and Morley in 1887 devised an experiment to measure the Earth's velocity relative to the aether and failed.

Aether and phlogiston are two of the many objects that now no longer exist within modern physics and yet they were real enough at the time of their use; effects were attributed to them and predictions about the physical world were successfully made with them.

It is clear from these observations that it is the theory, not the world, that defines and names the objects to be seen. It is the theory that gives meaning to an experiment. It is the theory which reflects the purpose of the theory-users since it marks what interests them. However, it is the world, not the theory, that allows us to support such theoretical objects through the observations (measurements) drawn from our experiments. We propose *distinctions* in the world as suggested by our theories and the world responds by *constraining* the range of values (measurements) of these distinctions. This is how hypotheses are supported or rejected and how knowledge of the world is obtained.

The relationship between the scientific activity of theory formation and experimental observations has a similar relationship to the human 'mind set' and perception; that is a person (say a domain expert) will tend to see what he/she expects or is trained to see. The process of theory formation or 'abduction' can be recognised to occur within human cognition at two distinct levels; 'automatic' and 'conscious'. Automatic abduction within vision leads to the emergent significance of certain features within the field of observation. Conscious abduction is the process by which the understanding of the world is formed and makes these features important. Thus, those individuals learning Arabic script or understanding X-ray plates have formed new perceptual responses by the end of their period of study.

The formation of a theory (abduction) and the support of a theory (induction) are distinct kinds of inference and both are different in one important respect to deduction (the use of a

theory) in that they are ampliative whereas deduction is not. Deduction is the process of reforming knowledge and is valid whereas both abduction and induction generate 'new' knowledge about the world (ampliative) and have no formal validation.

Any closed mechanical system (eg computers) that is used to model inference can only be deductive (or sub-deductive) since such a system will be at least formal, possibly consistent and at best valid. This is because:

a closed system is cut off from the world,

further knowledge of the world cannot be obtained,

valid operations are truth preserving,

operations that are not truth preserving will reduce the knowledge content through transformations,

and thus

such a system cannot be ampliative except by accident.

4. CENTRAL QUESTIONS

The same argument could equally well be applied to people so what is the difference? What are the limits of formal (ie mechanical) systems and why? Is it possible to produce a theory that clearly shows the necessary (and hence the minimum) components of systems with different capacities such as transformational (eg payroll program), learning (eg pattern recognition), intelligence (eg IQ problem solving), ampliative reasoning (eg induction), inventiveness (eg engineering), insight (eg abduction), etc; a theory that is both satisfactory and unambiguous. A satisfactory theory being one that is useful and an unambiguous theory being one that is testable.

5. THE GULF

The task of programming computers to perform computations, no matter how complex, presents no real difficulty. However, programming computers to interact with the world in a similar way to that of people is currently impossible. For example, during the decade of the sixties many of the problems in pattern recognition had been solved, however, there was one question in the activity that could not be answered and presented a barrier to further progress. The question was: "How do you choose the range of features (distinctions in the world) that are to be used to recognise the patterns?". Experience had shown that the choice of features could be critical in some cases but in others arbitrary choices were surprisingly successful (Fu 1968).

The root of the feature problem is that there is, in principle, an infinite set of possible features derivable from the real world. Any choice of features will restrict the information available for a pattern recognition or decision making calculation. The restriction occurs in two ways:

1 The raw information is limited to the measurements implied by the feature set.

2 The features will impose a structure on the incoming information (measurements) that will further restrict the available information and in some cases may introduce noise.

An example of this can be found in the features (eg edges, corners, parallel lines, etc) chosen to describe a fixed set of geometric structures (cubes, cylinders, spheres, cones etc). Although representations of badly defined or naturally occurring geometric objects can be analysed in terms of these basic features it will usually be very difficult to achieve good and consistent recognition (say better than 95%); a recognition rate that is appropriate for many practical applications.

There is another dimension to this problem which emerged from this work and remained a puzzle. The problem arose when the range of patterns was potentially infinite as might be found in the automatic description of camera images of scenes (an arrangement of geometric objects) or the interpretation of spoken sentences (a string of words). The question was "How do you construct a system that can interpret such complexes?" Part of this question was answered by developing a new class of pattern recognition which was essentially based upon discovering the "language" of the problem. Pattern recognition then became "sentence recognition" where scenes are described or spoken sentences interpreted. However, the pre-determination of the features or

words of the language will influence the simplicity of the grammar of the "language" that might emerge and in some cases this choice could be critical.

No automatic method was discovered to extract the "right" features from the domain under investigation; feature extraction and the subsequent syntax description were the province of the investigators (people). It was always possible to assess the information content of a feature (Fu 1968) but how do you choose the features in the first instance? This restriction made any attempts to simulate learning in a natural environment simply a rearrangement exercise and statements about "automatic induction" (ie abduction) were at best misleading.

The same problem of feature discovery arises today. This problem can be seen to occur in areas that range from the difficulty in correctly choosing the attributes for a company database to the problems of defining the primitives for knowledge representation in computers. One of the important domains currently under investigation is the discovery of the features in genetic material that can describe the dynamics of protein development and the gross structure of proteins from sequential information (Rawlings 1986).

A related problem occurs during certain classes of problem solving. Tasks such as scheduling or the development of proofs in non-simple domains usually require human intervention if they are to be successful. In Artificial Intelligence jargon these difficulties have been identified as the representation problem and the combinatorial explosion; they are difficulties that are often computationally intractable.

6. THE HUMAN BRIDGE

One of the solutions is not to attempt to solve the problem by computer alone but to incorporate the human as a component in the problem solving team (eg Morik 1988, Elleby et al 1988). This requires that the designer of such a cooperative system be aware of exactly how the problem solving task should be divided between the man and the machine. Unfortunately, we do not have any ready methods of designing mixed systems nor do we have a clear comparison of human and machine capability.

One of the clues towards resolving this problem is to be found in a refined description of inference. Inference is the basic mechanism that underlies all problem solving activities; it enables observations and hypotheses about the world to be used to solve problems in the world. Many logicians now recognise three classes of inference: Abduction, Deduction and Induction. Abduction is the creation of a hypothesis or the proposal of a solution path. Deduction is the process by which a hypothesis is used to make predictions or to perform the proposed solution. Induction is the assessment of the quality of a hypothesis or a solution.

It can be shown that the problem of discovering features for pattern recognition or primitives in knowledge representation or attributes for databases or functions for functional programming relies upon abduction as the principal mode of inference.

7. A RESEARCH APPROACH

It is clear from the above discussion that in order to begin to explain the puzzling phenomena, answer the central questions and bridge the gulf, a theory must be proposed that has the property of reducing uncertainty. Such a theory becomes testable through the process of assessing its capability to make predictions.

It is also clear from the above discussion that a theory can only be created around distinctions (eg features, measurements, results of tests) that are made in the world. To this end a simple taxonomy of knowledge will be proposed with which some bold statements will be expressed (a primitive theory). These bold statements will be a *priming* theory that starts a scientific activity. The priming theory will be tested against the world of systems; the results of these tests will be used to modify both the taxonomy and the statements. The full meaning of these statements will only emerge through experience and examples.

8. THE PRIMING THEORY

The theory relates knowledge to its representation in a system (eg a computer). It is the purpose of such a theory to provide an explanation of the limitations observed in using Artificial Intelligence to this end and to suggest a means by which these limitations may be overcome. The priming theory will consist of two groups of statements: Knowledge and Representation.

KNOWLEDGE

K1 **"The prime purpose of knowledge is survival."**

This statement (K1) is intended to prepare the ground for a definition of knowledge that is linked with successful actions.

Knowledge is generally considered in human terms; it would be easy just to claim that the human capacity for representing the World gives the race a selective advantage to the extent that actions are based upon the expectations that are derived from apposite knowledge and these expectations will lead to effective actions. In particular, the actions will improve opportunities for survival and reduce risk. However, I would like to extend the notion of knowledge beyond the human context so that we may be able to discuss the knowledge of animals, machines and possibly plants. This extension may, in some sense, devalue the meaning of knowledge by failing to capture the richness of our experience but it is hoped that the richness will be reinstated as a more complex manifestation of this simpler notion.

It could also be argued that knowledge, in any form, need have no teleological implications of 'the prime purpose'. However, it is difficult (if not impossible) to construct any descriptive system that is devoid of purpose and such a descriptive system represents the knowledge.

On the other hand, the purpose of knowledge might be something other than survival but it is difficult to suggest a purpose that does not ultimatly lead back to problems of existence of the users of knowledge or to a purpose that can provide a clear mechanism for justifying the form of the knowledge.

K2 **"The World is the source of all knowledge."**

Mathematical "knowledge" seems to be a counter example to this statement. However, the statement defines what is meant by knowledge. The World (or that much of the environment which can causally influence us) is that against which all knowledge (including mathematics) is measured and receives its justification. Knowledge that is derived through valid deductive inference is not new knowledge but the reformulation of existing knowledge (see below). On the other hand, the creation and exploration of an abstract world is knowledge of that abstract world and the activity is but a game unless it can be shown to have some point (or points) of contact with the World.

K3 **"There are three dimensions to knowledge."**

These dimensions are *Inference*, *Heuristics* and *Declarative Knowledge*. Inference is the knowledge of how to reason. Heuristics is the knowledge of how to apply the inference to the Declarative Knowledge to achieve a purpose. Declarative Knowledge comprises statements about the World; they are hypotheses.

Strictly, Declarative Knowledge is only one dimension of knowledge but common usage of the term "knowledge" will be recognised in this case.

K4 **"Declarative Knowledge reduces the uncertainty of the World."**

An example of the reduction of uncertainty is the observation of a sequence of numbers ranging from 0 to 9. Each number is observed to be equally probable ([probability of a number] = 0.1) and hence the information of each event is 3.3 bits (ie log(10) and is known as the *surprisal*). However, if you "know" that the sequence generated is Pi (or any stated sequence of numbers that is not necessarily generated by a mechanism) then every number is predictable (assuming no noise) and there is no uncertainty ([probability of a number/given that sequence is Pi] = 1). The information of each event is then zero bits.

K5 **"There are two classes of Declarative Knowledge (Hypotheses)."**

These classes are *Taxonomic* and *Transformational.*

Taxonomic Knowledge are the distinctions (range of values) or objects in the World. In the example of Pi (above) the Taxonomic Knowledge is the class of numbers that range from 0 to 9 and the class of numbers 0 and the class of numbers 1 and ..... etc. Note that a measure of information is always relative to a set of predefined distinctions; for example, the set of numbers. Taxonomic Knowledge, in this case, reduces the uncertainty from a possible infinity to 3.3 bits.

Transformational Knowledge can be considered as a collection of propositions that are predictive through the definition of constraints between distinctions. The Transformational Knowledge in the above example is the calculation of Pi given the previous sequence of numbers; it is a constraint that limits the possibility of the next number to one.

K6 **"There are no absolute distinctions of the World."**

Taxonomic Knowledge (ie the distinctions) can always be given structure. "Absolute" or "primary" distinctions of a representation depend only upon the purpose of the representation. Conversely, Transformational Knowledge can be reduced to Taxonomic Knowledge with a subsequent loss of detail; this is the process of *abstraction.*

Abstraction of detail is important and in most cases necessary for problem solving. For example, in the game of tic-tac-toe the distinction of the centre, corner and middle positions of the marks on the grid and the recognition of closed lines (where your opponent already has a mark) help describe in succinct and in general terms winning and drawing strategies. The only alternative would be to exhaustively enumerate all positions and best actions. However, the declarative knowledge in the form of corners etc can be described in terms of grid positions (declarative knowledge) with imposed constraints (taxonomic knowledge).

Successful actions (see K1) will depend upon the appropriate level of abstraction being completed. This would imply that the performance of any evolving (ie learning) system will be discontinuous over time. This result is supported by experiments described by Elliott Jaques et al (Jaques 1978).

An observation is an active not a passive event. Distinctions of the World are proposed through the process of abduction of type 1 (see below) and successfully applied through actions on the World. The actions may be complex involving the construction of artifacts or the learning of special skills.

K7 **"Declarative Knowledge need have no truth value."**

It is not of great significance that a hypothesis is true or false. Hypotheses are only more or less predictive, confirmed or useful (see Induction). A truth value is only a token associated with a set of hypotheses to mark their consistency and is used to show valid (ie deductive) argument. Different sets of hypotheses can be self consistent but inconsistent when placed together.

K8 **"There are two forms of Declarative Knowledge."**

Declarative knowledge has both an *intensional* and an *extensional* component. The intensional form expresses the criteria for belonging to a class and is equivalent to a *concept* (a generalisation). The extensional form is a set of examples (actual observations and measurements etc) of a given class.

These forms represent a **relationship** between the propositions that are the intensions and the propositions that are the instances of the extensions. For example, consider the propositions:

$$\text{ISA}(p,x), \quad p(x), \quad x$$

The intensional/extensional instance can be displayed as a layered structure as suggested by the following tables where layer 1 relates a single observation with its class, layer 2 is the level of predication of classes, and level 3 is the predication of predicates:

Layer	Intension	Instance
1	red	red
2	Colour(x)	Colour(red)
3	ISA(p,x)	ISA(Colour,red)

Some knowledge representations, such as logic and semantic nets, do not make this distinction; other representations, such as frames and database schemas, do. The distinction is important because it separates instances that support a hypothesis and the hypothesis itself.

K9 **"There are only three classes of inference."**

These classes are *Abduction*, *Induction* and *Deduction*. However, it is useful to recognise three types of abduction and three types of induction.

Abduction - Type 1 abduces new distinctions, type 2 abduces new hypotheses, type 3 abduces new heuristics.

Normally abduction is considered from the intensional point of view as a generator of hypotheses. However, abduction can be considered from an extensional point of view as being a set of hypotheses that are suitable for the purpose. There are two kinds of abduction sets: the *open* or infinite and the *closed* or finite.

Induction - Type 1 induces from samples and is based upon statistical (eg observed frequencies) considerations, type 2 induces from continuous observation and is based upon confidence, type 3 induces from utility and assesses the worth of a hypothesis.

Further structuring of inference is possible (see Addis 1988).

REPRESENTATION

R1 **"A *representation* involves three elements: the signs, the symbols and the interpreter."**

The signs are the concepts (objects and relationships) that have been made distinct and mapped onto a set of symbols. Both the process of making the distinctions and the mapping are made by the interpreter (usually a person since it is difficult to imagine a representation that does not involve human cognition).

R2 **"A *system* and the knowledge to be represented must be homomorphic."**

Any system, be it organic, inorganic, natural or artificial, that is used to represent knowledge must be structured to include representations of the three dimensions of knowledge.

Representations may be *explicit* or *implicit*. Explicit representations are distinctions within the system that are related directly to the knowledge represented. Implicit representations are implied through the structure (see R4).

R3 **"A system is part of the World."**

Symbols are distinctions (features) of the system that have been selected to represent the Taxonomic Knowledge. The state of the system is defined in terms of its distinctions and the constraints on the states of the system represent the Transformational Knowledge. Inference is *activated* through the changes in state of an *active* representation under the control of Heuristics. Representations that cannot be activated without an external agent (eg script) are *passive*.

R4 **"*Deduction* is the only inference that can be activated in a system."**

Deduction is taken to mean the mechanical component of inference (eg symbol manipulation) such as Modus Ponens or its generalization resolution. It does not involve the *heuristic* component that selects and guides the activity to achieve deduction in the full sense (see K3). This is the traditional Artificial Intelligence view (Addis 1985).

Knowledge of the World will either remain constant (although possibly reformed) or reduce in a closed system. A closed system is one that is isolated in terms of information and is cut off from the World.

Abduction, Induction and Heuristics are the emergent properties of a deductive process in a higher order system.

A higher order system is a system whose domain includes the primitive system (the primitive system has the domain knowledge for the initial objective of the total system). Such a system will be called *Reflective* (ie it can reflect on its own knowledge).

R5 **"The *structure* of a system is defined by the represented knowledge."**

The structure of a system is the set of *distinctions* (features) and the set of *constraints* imposed on the possible combination of values of those distinctions (see above).

A *constraint* is a restriction upon possibilities and hence provides a means through which to make predictions. Knowledge is represented in a system by its structure. It is for this reason that notations that are specifically designed to describe constraints, such as grammars or networks, are chosen to represent knowledge. However, notations designed for specifying machine structures (eg programming languages) will be limited to representing only certain kinds of knowledge.

The *Domain* of a system is the subject matter of the Declarative Knowledge represented by the system's structure.

R6 **"The minimum requirement (sufficient but not necessary condition) for a system to be *Intelligent* is that it must be able to demonstrate all three classes of inference."**

This will be considered to be a definition of intelligence but could be rejected if we discover a system that is considered to be intelligent by any normal standards which does not show the three classes of inference. It is assumed that the abduction is "open" but a version of intelligence might be considered that is "closed".

Intelligence tests can be considered to be solved by the application of these three classes of inference. Thus, a series with a missing element will stimulate an abduced hypothesis that is employed to deduce the missing element and then the hypothesis is induced (ie supported) through critical analysis. Low performance at intelligent tests may be due to an inadequate capability in any of these three steps. High performance at intelligent tests would primarily be due to the ability to abduce just a few possible hypotheses of the right kind.

Human abduction is dependent upon an evolutional predisposition and an educated understanding of the World (cf K1).

R7 **"The minimum requirement for a system to display *Learning* is that it must be reflective."**

This follows from K9 and R4.

It will be the purpose of the research programme to examine many different systems and to test, refine and expand the above statements through the normal scientific process. The above statements are **not** the final theory; they are a starting point and at best demonstrate that a theory on such a subject is conceivable.

9. POSSIBLE CONSEQUENCES

The priming theory is intended to focus the attention of investigators onto specific considerations and provide a language for discussion. An explanation of some of the puzzling observations and the questions posed at the beginning of this paper can now be suggested but it will remain for a more detailed analysis to confirm (or otherwise) these explanations.

The theory has given structure to knowledge and this structure will help to describe the wide range of different manifestations of knowledge. It is now possible to characterise representations, to show how these representations relate to different components of knowledge and to know what can be achieved with specified systems. Most computer systems have a core of active representations in the form of a program and an interface to the user that is passive (eg predefined messages that need interpretation). Many programs are single level and do not have the capacity to display ampliative reasoning of any kind. The main difference between expert systems and other com-

puter programs is that expert systems contain asserted knowledge that is explicit. **Designs of explicit multi-level systems would provide a useful model of this aspect of the theory.**

Ampliative inference (ie abduction and induction) is a puzzle since given any system that is deductive it would seem that for any situation, no matter how simple, there is the possibility of proposing an infinite number of potential hypotheses (cf R4). The number is greatly reduced if the system has a predisposition to generate only plausible hypotheses (which seems a tautology). Such a predisposition means that the World can be characterised in terms of higher order knowledge and the World is restricted to a clear set of possibilities. This would suggest that there is a grammar for hypothesis construction. It is a grammar that reflects the World and has been formed under the pressure of evolution (cf K1). Further, it sets limits on what may be abduced. **It would be a major research objective to determine this grammar.**

Central to a theory of knowledge is the idea of a system. The capacity of any system to represent knowledge depends upon the available features and structures of the system (cf R5). In particular, we should not expect to see biologically represented knowledge confined totally to neurophysiological systems. All features of a system are open to exploitation (cf R3). An extreme example of a passive biological representation is the rhubarb plant which can be considered to contain a specialised representation of irrigation within the shape of its leaves and stalks. This representation forms an active system when it rains.

The natural features of certain artificial systems will restrict their use to representing specific types of knowledge (cf R2). It is the role of the designer/engineer to employ these systems in practice and this needs insight. In some cases, where there is a well established class of artificial systems (eg computers), techniques have evolved to limit the perception of the designers/engineers to the class of knowledge that is most efficiently represented by that system. This restriction on perception is usually achieved by limiting the given notation.

An example of a technique is relational analysis. Relational analysis is specifically focused on taxonomic knowledge. The technique ensures that all but the most primitive of constraints (ie transformational knowledge in the form of functional dependencies between key and own attributes within normalized relations) are eliminated from the final specification. The reason for this taxonomic orientation is that it reflects the representational capacity of mass computer storage. **It would be another research objective to develop a formal association between machine features and design notation.**

Other design criteria orientated towards incorporating people into a system can be proposed. In particular, if systems are limited such that they do not have higher orders of knowledge (ie knowledge about knowledge) then they cannot show abductive behaviour and thus any class of problem that involves insight or judgement will require human participation (cf R4). Also the limit of the detail necessary to be represented is an issue that sets a horizon on the distinctions to be made. This horizon defines the appropriate primitives for the task domain and this horizon depends upon what active component the users can supply to the primitives to complete the system (cf K6). **A further research objective would be to show the relationship between passive representation and active human inference.**

A theory of knowledge should indicate what are the necessary elements to be represented in order to achieve a specific activity and should show clearly what cannot be achieved with certain techniques. The statement that there are no absolute distinctions (cf K6) suggests that knowledge will evolve in layers of abstractions. Examples can be found in artificial vision systems that start with pixels to form features that are, in turn, used to specify scenes. The layering is a result of a Taxonomic, Transformational, Taxonomic cycle. Such layering would be expected to be observed in naturally occurring systems. **Investigations of artificial and natural systems in these terms would help develop the theory.**

The theory should offer a means through which all systems can be judged or designed: from the simplest computer program to the most advanced chess playing program; from the switch to the neural net. **The theory of knowledge should lead towards the provision of a tool with which the representation capacity of a machine can be analyzed and from this analysis, generate a domain analysis method that matches.**

Alternatively, the theory should suggest an analysis of a domain and propose a machine that is appropriate for the purpose. **On a grander scale, the theory could offer the neurophysiologists a framework within which they could interpret their findings.**

10. ACKNOWLEDGEMENTS

I would like to thank my students and colleagues for the many interesting conversations about knowledge and knowledge-based systems. In particular, I would like to acknowledge the contributions made by Mark Nowell to the ideas and insights that have emerged in this paper. My guide through the works of Pierce and other philosophers has been Dr Jim Tiles. However, the interpretation and any errors are mine alone.

11. REFERENCES

1. Addis, T. R. (1985) "Designing Knowledge-Based Systems" pub Kogan Page, London, UK.
2 Addis, T. R. (1987) "A Framework for Knowledge Elicitation.", Proceedings of the first European Workshop on Knowledge Acquisition for Knowledge-Based Systems, Reading University, ppA1/1-A1/16.
3. Addis, T. R. (1988) "A Knowledge Organisation for Abduction.", Proceedings of The Interdisciplinary Information Technology Conference, Bradford University, pp19.
4. Charniak, E. (1988) "Motivation analysis, abductive unification and nonmonotonic equality" Artificial Intelligence 24, pp275-295.
5. Charniak, E. and McDermott, D. (1985) "Introduction to Artificial Intelligence", Addison-Wesley.
6. Elleby, P., Fargher, H. E. and Addis, T. R. (1988) "Reactive Constraint-Based Job-Shop Scheduling." Proceedings of the 2nd International Conference on Expert Systems and the Leading Edge in Production Planning and Control, Charleston SC, USA, pub North Holland, pp 1-10.
7 Emde, W. (1987) "Non-cumulative Learning in Metaxa.3." IJCAI-87, pp208-210, Morgan Kaufmann.
8. Fu, K. S. (1968) "Sequential Methods in Pattern Recognition and Machine Learning.", Academic Press, London and New York.
9. Jaques, E. (1978) "Levels of Abstraction in Logic and Human Actions." pub Heinemann, London.
10. Johnson, L and Keravnou, E. T. (1985) "Expert Systems Technology - a guide." Abacus Press, London.
11. Johnson, N (1987) "Mediating Representations in Knowledge Elicitation." Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems. ppA2/1-A210.
12. Kelly, G. A. (1955) "The Psychology of Personal Constructs." Norton. NY.
13. Langley, P. et al (1987) "Scientific Discovery", MIT Press, London.
14. Langley, P. (1988) "Invited Talk", EWSL88, European Workshop Session on Learning, Turing Institute, Glasgow.
15. Nilson, N. J. (1980) "Principles of Artificial Intelligence" Tioga Pub, California.
16. Morik, K. (1988) "Sloppy Modeling" Technical Report, Technical University Berlin.
17. Peirce, C. S. (1958) "Science and Philosophy." Collected papers of Charles S. Peirce 7, Harvard University Press.
18. Rawlings, C. J. (1986) "Representing Protein Structures in PROLOG: the PROLOG representation.", Imperial Cancer Research Biomedical Computing Unit. Submitted as part of results of SERC Contract No: SO/351/84.

THE VALIDATION OF KNOWLEDGE ACQUISITION: METHODOLOGY AND TECHNIQUES

by

Izak Benbasat and Jasbir S. Dhaliwal
Faculty of Commerce and Business Administration
University of British Columbia
2053 Main Mall
Vancouver, B.C. Canada V6T 1Y8
(604) 224-8396

1. INTRODUCTION

With the recent proliferation of knowledge-based systems (KBS) and the attention paid to their development, an increasing number of authors have expressed interest in methods for the validation of these systems (O'Leary, 1987; Shaw and Woodward, 1988). According to O'Keefe et al. (1987) validation of KBS has mostly been handled in an *ad hoc*, informal way. In addition to the concern about properly testing systems as to their value and correctness, there is the need to be able to compare and evaluate the multitude of knowledge acquisition (KA) methods and tools (both manual and automated).

Similar concerns were raised at the 3rd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop held at Banff, Alberta in November 1988. Several of the participants referred to issues associated with validation and evaluation (Enand et al., 1988; Gaines, 1988; Johnson et al., 1988; Shaw and Gaines, 1988), though each examined only one or a few of the aspects of the overall problem. In a paper discussed at the Banff Workshop, entitled "A Framework for the Validation of Knowledge Acquisition" (Benbasat and Dhaliwal, 1988), we attempted to provide a comprehensive conceptual framework and a set of definitions in order to clarify the different issues associated with validity and evaluation. This current paper extends our earlier work by discussing how the validation process could actually be conducted, how various researchers and knowledge-based system developers have approached this issue, and what techniques adopted from other fields such as simulation, modelling, and software development could be utilized in this respect.

The next two sections of the the paper utilize the definitions and the life cycle based validation framework from our earlier paper to discuss the validation of knowledge acquisition. Section 4 describes and provides examples of specific KBS validation techniques and procedures to assess the validity of the KA process and of the knowledge acquired. Finally, section 5 examines some of the factors that need to be considered in the selection of such techniques and procedures.

2. DEFINITIONS

This section presents a precise definition of validity and contrasts it with other related terms, such as verification and evaluation, with which validity is sometimes used interchangeably.

In psychometrics validity is the degree to which a measuring instrument actually does what it is intended to do (Nunnally, 1967, p. 65). Consequently, if the objective of the KA process is to capture the knowledge of an expertise source then *the validity of the KA process is the degree of homomorphism between the representation system, i.e., KBS, and the system that it is supposed to represent, i.e., the expertise source* (Vandierendock, 1975). This representational homomorphism definition of validity is consistent with Bobrow's (1975) notion that at an abstract level a representation can be viewed as "the result of a selective mapping of aspects of the world." A world-state (expert source) can be modelled into a knowledge-state (KBS) and questions about the world-state can be answered either by observing the world-state or by questioning the corresponding knowledge-state.

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

Evaluation is different from validation. It is possible that the KA process could fully capture the knowledge of a source expert, while at the same time what has been captured, i.e., the expertise, could fall short of a given quality level as defined by another set of experts. There are several of types of evaluation, as distinct from validity, that will be referred to in this paper:

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

3. LIFE CYCLE BASED VALIDATION OF KBS

Substantiation of the validity of a knowledge-base (KB) involves undertaking tests and evaluations *within the different stages* of the development process. Figure 1 at the end of this paper depicts a life cycle model of the KBS development process. It specifically isolates, at different stages of the life cycle, three progressively more developed knowledge-bases, all of which require different combinations of validation criteria, measurements and techniques. It is important to note that while the life cycle is presented in a somewhat static sequential manner in the interest of clarity of presentation, in reality the different phases overlap significantly, and there are multiple iterations of each phase.

FIGURE 1 HERE

The conceptual knowledge-base represents an abstraction of the source knowledge-base in the real world. It can be expressed in the form of words, diagrams, schematics, or by mathematical representations (Oren et al., 1985). *Conceptual validation* assures that the assumptions and theories underlying the conceptual knowledge-base are correct and that it is a representation of reality that is reasonable for the intended application.

Elicitation validation provides assurance regarding the completeness and correctness of the *KA process* by which the conceptual knowledge-base is instantiated and extended into an elicited knowledge-base.

Implementation validation is aimed at ensuring the quality of the translation process that transforms the elicited knowledge-base into a functional KBS (i.e., a working system or prototype). It is analogous to *verification* in traditional software engineering.

A functional KBS can further be assessed in two ways: functional evaluation and representational validation. *Functional Evaluation* resembles the common interpretation of KBS evaluation and refers to the evaluations of performance quality, utility, and usability as described in the last section. *Representational validation* attests to the match between the characteristics of the implemented KBS and those of the expert source. This validation takes two forms: 1) *structural matching* which refers to the validation of knowledge representation and processing (reasoning) logic, and 2) *behavioral matching* which is the similarity between the output of the KBS and that of the source expert(s) (i.e., the expert or experts on which the KBS is modelled). Such assessments can also include comparisons to alternative nonsource expert knowledge-bases existing in reality, and/or to idealized knowledge-bases framed in the minds of the system developers or users.

4. VALIDITY ASSESSMENT TECHNIQUES AND PROCEDURES

Figure 1 comprises three progressively more developed knowledge-bases. Tables 1, 2 and 3 (all tables are located at the end of the paper) define and provide examples of the types of assessment tests necessary for validating and evaluating these knowledge-bases representing the different stages of KBS development. A detailed explanation of these tables can be found in Benbasat and Dhaliwal (1988). Furthermore, considering that validation is the process by which assessment tests are undertaken to gather a web of evidence pertaining to the validity of a KBS, Table 4 defines the different types of evidence that could be collected and provides a summary of the means of collecting such evidence.

TABLE 1 HERE

4.1 Conceptual Knowledge-Base Assessment Methods

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

4.1.1 Assessment of Conceptual Specifications - Woodward (1988) has developed a semi-automated tool (Cognosys) and has proposed a structured methodology for domain definition and organization at the pre-KA stage to build conceptual domain models that serve to guide subsequent KA activity. Such domain models facilitate conceptual KB qualification, validity and formal checking (see Table 1). Additionally, they act as comparison standards for both structural validity and verification tests during the subsequent stage of elicited KB validation (see Table 2).

At this early stage of the development process, the validity methods are mainly based on collecting *content-related* evidence. The adequacy of content can be evaluated by methods such as a "structured walkthrough" which essentially involves bringing together a group of people to examine the contents of a representation of the KB. The intent is to find errors, inaccuracies, omissions, contradictions, and redundancies.

A suggested technique for knowledge-base qualification is the use of structured walkthroughs involving both source and nonsource experts, knowledge engineers (KE), and prospective users of the KBS. Prospective user involvement ensures that the correct system is specified for the problem to be solved. The purpose here is to examine modelling documentation such as decision trees and knowledge hierarchy charts to isolate problems, inaccuracies, ambiguities, and omissions in the conceptual KB. For example, Johnson et al. (1988) describe the use of the Systemic Grammar Network -- a computer independent "mediating representation" notation -- as a documentation and communication aid during the conceptualization stages of early KA.

Knowledge-base validity invokes comparisons between the specifications and the real source of expertise based on a set of structural criteria. These are similar to structural assessment criteria for the elicited KB and are discussed below. At this stage the most reasonable validity check would be for the KE and expert to go over the representation of the conceptual KB. For example, Johnson et al. (1988) suggest that the KE check the terms used in the representation with the expert. Other secondary techniques that are applicable to assessing conceptual KB validity, especially if the expertise is being gathered from reconstructed sources, include assumptions challenging and seeking convergent and divergent validity from related theories and practical observations in the application domain.

4.1.2 Assessment of Test-Cases - Test-cases need to be assessed with respect to the problem domain (test-case applicability) and to the source of expertise being modelled (test-case validity). This is important as nonrepresentative and unrealistic test-cases can lead to incomplete and erroneous knowledge acquisition. This is especially true in domains such as law, where a finite number of legal cases define

the precedents. Such cases can be used both to induce a scheme to represent the domain as well as to later test its validity (Chan and Benbasat, 1989). Ruberg et al. (1988) describe the use of both manual editing by a domain expert and an automated logical consistency verification system to validate rules induced by two machine learning algorithms from a set of 500 example cases.

4.2 Elicited Knowledge-Base Assessment

Starting with the conceptual specifications, the KA process produces an *elicited knowledge-base in the* form of a 'complete' expertise specification formalized to some selected knowledge representation (for example, rules, frames, or semantic nets) but in nonmachine-readable form. The methods and procedures used to validate elicited knowledge-bases are outlined in Table 2.

TABLE 2 HERE

Construct-related evidence of validity, which is not always easy to obtain and specify, is of highest priority in validating the elicited KB. It is collected by first developing an operational definition of the construct (i.e., the KB) and then developing hypotheses about how this measure relates to other measurable variables. Note that a definition of the construct is now available because a set of formalized specifications, i.e., the elicited KB, has been developed. Next, a conceptual framework should be established to show how the construct relates to and is distinguished from other constructs and measures associated with it. Convergent and divergent evidence, as specified by the conceptual framework established, could then be sought. Shaw and Woodward (1988) provide examples of such evidence. They mention a category labelled objective validity which measures the degree to which the facts in the KB agree with well established facts in the domain. This is an example of convergent evidence. On the other hand, conflict with the predictions of a theory would constitute divergent evidence. Comparing the KBs generated using the same KA method by two or more individuals with similar levels of expertise would be helpful for gathering convergent and divergent evidence as to the validity of a KA tool. Several examples of this procedure will be discussed in section 4.2.1.

4.2.1 Assessment of Structure - Assessments of structure have two aspects: comparisons to the source expert (**structural validity**) and to other expertise sources (**structural comparison**). Note that an examination of the data collection and experimentation methods required to make such comparisons (**data collection veracity**) is also required to account for the validity and reliability of the data collected (for example, variances arising from expert fatigue, motivation, and communication problems in test data collection).

Specific assessment techniques include comparing the objects and relations in the elicited KB to those of both source and nonsource experts. Shaw and Woodward (1988) accomplish this by having source experts rate their own entailments as generated using the *KSS0* system (structural validity), as well as rating the entities and attributes generated by other experts (structural comparison). O'Leary (1988) calls these *intra-expert* and *inter-expert* tests, respectively. Further, Shaw and Gaines (1988) have developed a methodology and classification scheme to measure the degree of structural similarity, based on attribute names and attribute values for agreed upon entities, between two or more experts (structural comparison). If experts agree in both terminology and concepts when referring to a common set of entities this is termed as *consensus*, whereas disagreement in both is called *conflict. Correspondence* refers to agreement solely only on concepts, and *contrast* to agreement on only terminology. This scheme represents a use of convergent and divergent evidence to test the construct validity of the KB (structural comparison). Another technique is to compare the number of values an attribute or property can take in the elicited model as compared to the source expert. This is a test for range correspondence and attests to both the accuracy and exhaustiveness of variable ranges.

Besides such analysis of the knowledge-base, it is also vital to validate aspects of inference in the KB. Procedural traces and snapshots obtained from the source expert

using protocol analysis or structured walkthroughs could be compared to the processing steps in the elicited KB to determine the operational or event correspondence between the two structures. This will reveal differences, if any, in processing strategies between the two.

4.2.2 Assessment of Behavior - Assessment of the behavior of the elicited KB is less useful than the assessment of its structure. It is performed with respect to two criteria: source expert behavior (**behavioral validation**), and other expert behavior (**behavioral comparison**). Although the KB is not yet in an operational form, and criterion-related evidence cannot be easily collected, it is still possible to logically validate the behavior of segments of the elicited KB prior to implementation. This can be done by collecting evidence through techniques such as structured walkthroughs (Yourdon, 1986) by both experts and KE. For example, missing rules can be detected by logically tracing back incorrect conclusions. The earlier assessed test-cases should prove useful for this purpose. Chan and Benbasat (1989) report on the use of both real and hypothetical cases by a KE to test the expert's categorization scheme in the development of a KBS in the domain of law. Johnson et al. (1988) suggest the use of real cases to test the validity of the elicited KB (behavioral comparison) as well as asking the expert to comment on the "chunks" of the KB as developed by the KE (behavioral validation). They also involve a wider population of experts (behavioral comparison) and obtain their comments either through intensive interviews or using a formal questionnaire. Walkthroughs with hypothetical cases could also used to compare the knowledge of other experts to that of the source expert.

O'Leary (1988) maintains that ensuring the completeness of a KB is a more onerous issue in validating the elicited KB than ensuring KB accuracy, and proposes the use of contingency tables as a technique for ensuring completeness. Manual sensitivity analysis could also be used, for besides providing the KE with assessment feedback, it can guide ongoing KA. However, in general it is best to concentrate on the assessment of behavior after an *operational* KBS is available.

4.2.3 Elicited KB Verification - As reflected in Table 2, assessment of the elicited knowledge-base includes two verification type tests: **verification** against the conceptual KB specifications and verification against the specifications of the elicitation formalism used (**formal checking**). Verification would be facilitated by the existence of a formal model created at the conceptual KB development stage, for example by a tool such as Cognosys (Woodward, 1988). Formal checking would be expedited if the language used to represent the elicited KB adhered to well defined rules, such as the Systematic Grammar Network utilized by Johnson et al. (1988). Note that in the case of automated KA systems such as ACQUINAS (Boose and Bradshaw, 1987) or KSS0 (Gaines, 1987), the elicited KB by definition adheres to the specifications of the elicitation formalism used.

4.3 Implemented Knowledge-Based System Assessment

The techniques and procedures that can be used to validate an implemented KBS are outlined with examples in Table 3.

TABLE 3 HERE

Criterion-based evidence of validity which was difficult to collect previously, since there was no operational KBS to provide output (behavior) that could be compared to other expert behavior, is of priority at this phase. Such evidence can be collected in three ways: predictive, concurrent, and post-dictive. *Predictive* refers to the situation where criterion data is collected in the future. Turing tests might involve the *concurrent* collection of criterion data from the KBS and the human expert. *Post-dictive* refers to the case where the criterion data has been collected in the past, such as comparing KBS output against test-cases with known outcomes.

Such validity evidence could be collected from the source expert, other nonsource experts, both realistic and hypothetical test-cases, and from the predictions of theory.

In addition, the way criterion data is collected might often be a source of error. A discrepancy between the construct (KBS) and the criterion evidence might be due to a real mismatch or incorrect measurement of either the criterion or the construct, or both. This issue is considered under data collection and experimentation *method* assessment in Table 3.

4.3.1 Implemented KBS Verification - The system construction process translates the elicited expertise specification resulting from the KA process into machine executable form. Verification tests are therefore necessary to ascertain the correctness of this translation process. Specific techniques here include code inspection, code walkthroughs, logical checking, rule execution tabulation, automated integrity enforcement, use of (rule) templates, and other assessments relating to traditional software quality standards. Along these lines, Nazareth (1989) notes that errors normally encountered in the development of rule-based systems take the form of redundancy, conflict, circularity and incompleteness, and presents a series of strategies for their detection.

4.3.2 Assessment of Structure - Structural assessment of the KBS utilizes similar techniques and criteria to that of the elicited KB. Comparisons need to be made to both the source expert (structural validity) and to other existing nonsource expert sources (structural comparison).

Considering that the same structural assessment techniques can be applied to validating both the elicited KB and the implemented KBS, more comprehensive and stringent assessment at the elicited KB stage can reduce the need for a similar assessment at the implemented stage. It is more advisable at this stage to emphasize the assessment of behavior.

4.3.3 Assessment of Behavior - Behavior assessment techniques like sensitivity analysis, volume tests, degenerate tests, extreme-value tests, and model-component tests (Sargent, 1985) take on increased importance here over content-related validity techniques like the logical walkthroughs conducted to assess behavior at the elicited KB validation stage. In addition to output behavior, the reasoning process by which the output is generated could also be evaluated. Examples of such comparison criteria might be the number of attributes and/or alternatives considered, the number of iterations the system goes through, the breadth or depth of search, and event or operational correspondence. This is because now a functional KBS, or a prototype of it at least, can be used and criterion-related evidence collected. The resulting outputs from such tests can be compared to outputs generated by both the source expert (behavioral validity) and other nonsource experts (behavioral comparison). The latter is necessary to reveal blind spots between the source experts and knowledge engineers caused by hidden assumptions. Weitzel and Kershberg (1989) argue that someone other than the source expert, with different hidden assumptions, should be involved in such a validation procedure.

Thus at this implemented KBS stage, not only is behavioral assessment more important, it can also provide clues as to event correspondence aspects of structural validity as well as defects in implementation (verification). Enand et al. (1988) provide an example of this, in relation to validating large diagnostic knowledge-bases, by including in their methodology a separate stage to ensure that the procedure taken by a KBS is correct in terms of output (behavior). This also highlights the fact that validity assessments should be performed as a coordinated and integrated plan of action. Geissman and Schultz (1988) suggest that such validation testing should begin early, even when only a subset of functions has been implemented, and propose regression testing and parallel testing over time as techniques.

4.3.4 Assessment of Usability - Two major types of assessment are necessary here. Utilization involves contrasting the frequency of use of the KBS against the use of the expert (usage comparison). Interface Validity refers to the ease of use and understandability of the KBS interface. Data for making interface comparisons is obtained by measuring users' perceptions. As reflected in Table 3, both these assessments are made by comparison to the "interaction with a source expert" (i.e., validity). The importance of usability validation procedures is evident from Ribar's (1988) validation of a KBS developed to aid in the analysis of commercial loan

collectibility. It was found while the KBS produced judgements similar to that of the source expert when used by the same expert, it produced inconsistent results in the hands of other users. Thus, while it did not misrepresent the expert's thinking, it was not sufficiently representative or valid to be usable. Kitto (1988) had a similar problem when evaluating the interface of the KRITON tool for KA. Furthermore, Chan and Benbasat (1989) observed that assessments of KBS interface and usability take on increased importance when organizational funding and continuation of the KBS development project are highly dependent on the support of top management to whom the KBS is demonstrated.

5. CONSIDERATIONS FOR THE SELECTION OF VALIDATION TESTS AND PROCEDURES

The validation methodology presented in this paper contains a wide array of detailed and complicated validation tests and procedures. It is evident that these involve significant cost and effort on the part of a variety of people, such as experts, knowledge engineers and users. The decision of how much effort to apply to the validation process and which aspects of the life cycle to validate depends on a host of considerations.

An important factor is the determination of the benefits accruing from a particular portfolio of validation tests and the cost of conducting these tests. In Table 5 we present a consequence profile that could be applied to aid such a cost-benefit determination. It highlights the need to measure and consider both the positive and negative consequences of KBS performance in cost-benefit analysis. For example, in the context of diagnostic KBS in the medical and nuclear waste management domains, it would be crucial to determine the consequences of having either a false-positive or a false-negative scenario prior to appraising and selecting validation procedures. On the other hand, a false-positive in a KBS dealing with credit granting decisions to individual lenders would not obviously have the same consequences. Thus, a managerial decision is required as to the effort to be expended on validation and the correct selection of validation techniques and procedures, based on the consequences of erroneous diagnosis and the costs of the test. Along these lines, O'Leary (1988) proposes that validation should focus on examining those rules or frames that have costly consequences, those with high certainty factors, and those with a greater frequency of use.

		Actual Condition: Positive	Actual Condition: Negative
Predicted Condition	Positive	Sensitivity	False-positive
	Negative	False-negative	Specificity

Table 5: Consequence Profile for Selecting Validation Tests

Dhaliwal and Benbasat (1989) have suggested a model of the factors influencing knowledge acquisition activity. These factors include the characteristics of the task domain, the KE, the source expert, and the organization supporting the KBS project. It is likely that the risks associated with the KA process would be higher under some conditions, such as the lack of previous experience in KBS development on the part of

the organization, the experts and the KE. Clearly, the importance of extensive validation would be more important when such risks are higher. Similarly, if the KBS is important for the operational viability and success of the organization, then validation tests should be more extensive and stringent.

Another related concern that warrants attention is the overall objective of the validation undertaken. A distinction needs to be drawn between the testing of KBS performance systems and the validation of expertise captured in a knowledge-base. For example, the Xie et al. (1988) and the Enand et al. (1988) studies are more focused on the testing of functional KBS, while the Shaw and Woodward (1988) and the Shaw and Gaines (1988) studies emphasize knowledge-base (expertise) validation in relation to a KA tool. Testing is a more functional version of validation and represents an effective means of ongoing knowledge acquisition. The validation of expertise acquired meanwhile is vital in assessing the value of particular validation techniques or procedures.

6. CONCLUDING COMMENTS

The focus of this paper was the central role validation should play in the development of knowledge-based systems. Recently there has been increasing attention paid to this issue in the literature. For example, Enand et al. (1988) assert that KA is not complete until a KB is fully verified and validated, and that validation seems critical as KBS approach deployment in the field. Woodward's (1988) variant of the KBS development process explicitly views validation/verification as an integral part of knowledge acquisition. These opinions concur with our view of validation as being a component subactivity of knowledge acquisition. In KBS development it is inadequate to relegate validation to being a distinctly independent and separate activity that is to be considered only subsequent to the completion of all knowledge acquisition and implementation. Rather, it is a key aspect of the initial modelling, detailed elicitation and implementation of KBS. Additionally, it can reasonably be argued that validation is the process of knowledge-base refinement and that during this process KBS are substantially enhanced and expanded. The task ahead for KA researchers is to develop and improve the specific methodology and techniques that could be applied to this important activity.

ACKNOWLEDGEMENTS

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

REFERENCES

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

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

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

Benbasat, I., and Dhaliwal, J.S. (1988). A Framework for the Validation of Knowledge Acquisition, *Proceedings of the 3rd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, 1988, pp. 1.1-1.18.

Chan, C., and Benbasat, I. (1989). Knowledge Acquisition: Descriptions and Prescriptions Based on Three Case Studies, Working Paper No. 89-MIS-008, Faculty of Commerce and Business Administration, University of British Columbia, 1989.

Dhaliwal, J.S., and Benbasat, I. (1989). A Model for the Empirical Investigation of Knowledge Acquisition Techniques, *Proceedings of the 2nd International IFIP/IFAC/IFORS Workshop on Artificial Intelligence in Economics and Management,* Singapore, January, 1989.

Enand, R., Kahn, G.S., and Mills, R.A. (1988). A Methodology for Validating Large Knowledge-Bases, *Proceedings of the 3rd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, 1988, pp. 5.1-5.11.

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

Gaines, B.R. (1988). Knowledge Acquisition and Technology, *Proceedings of the 3rd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, 1988, pp. 8.1-8.21.

Geissman, J.R., and Schultz, R.D. (1988). Verification and Validation of Expert Systems, *AI Expert*, Vol. 3(2), February, 1988, pp. 26-33.

Johnson, N.E., Tomlinson, C.M., and Johnson, L. (1988). Knowledge Representation for Knowledge Elicitation, *Proceedings of the 3rd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, 1988, pp. 13.1-13.15.

Kitto, C.M. (1988). Progress in Automated Knowledge Acquisition Tools: How Close Are We to Replacing the Knowledge Engineer?, *Proceedings of the 3rd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, 1988, pp. 14.1-14.13.

Nazareth, D.L. (1989). Issues in the Verification of Knowledge in Rule-Based Systems, *International Journal of Man-Machine Studies*, Vol. 30, pp. 255-271.

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

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

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

O'Leary, D.E. (1988). Methods of Validating Expert Systems, *Interfaces*, Vol. 18(6), November-December, 1988, pp. 72-79.

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

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

Ribar, G.S. (1988). Expert Systems Validation: A Case Study, *Expert Systems Review,* Vol. 1(3), pp. 26-28.

Ruberg, K., Cornick, S.M., and James, K.A. (1988). House Calls : Building and Maintaining a Diagnostic Rule-Base, *Proceedings of the 3rd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, 1988, pp. 28.1-28.20.

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

Shaw, M.L. and Gaines, B.R. (1988). A Methodology for Recognizing Conflict, Correspondence, Consensus, and Contrast in a Knowledge Acquisition System, *Proceedings of the 3rd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, 1988, pp. 30.1-30.19.

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

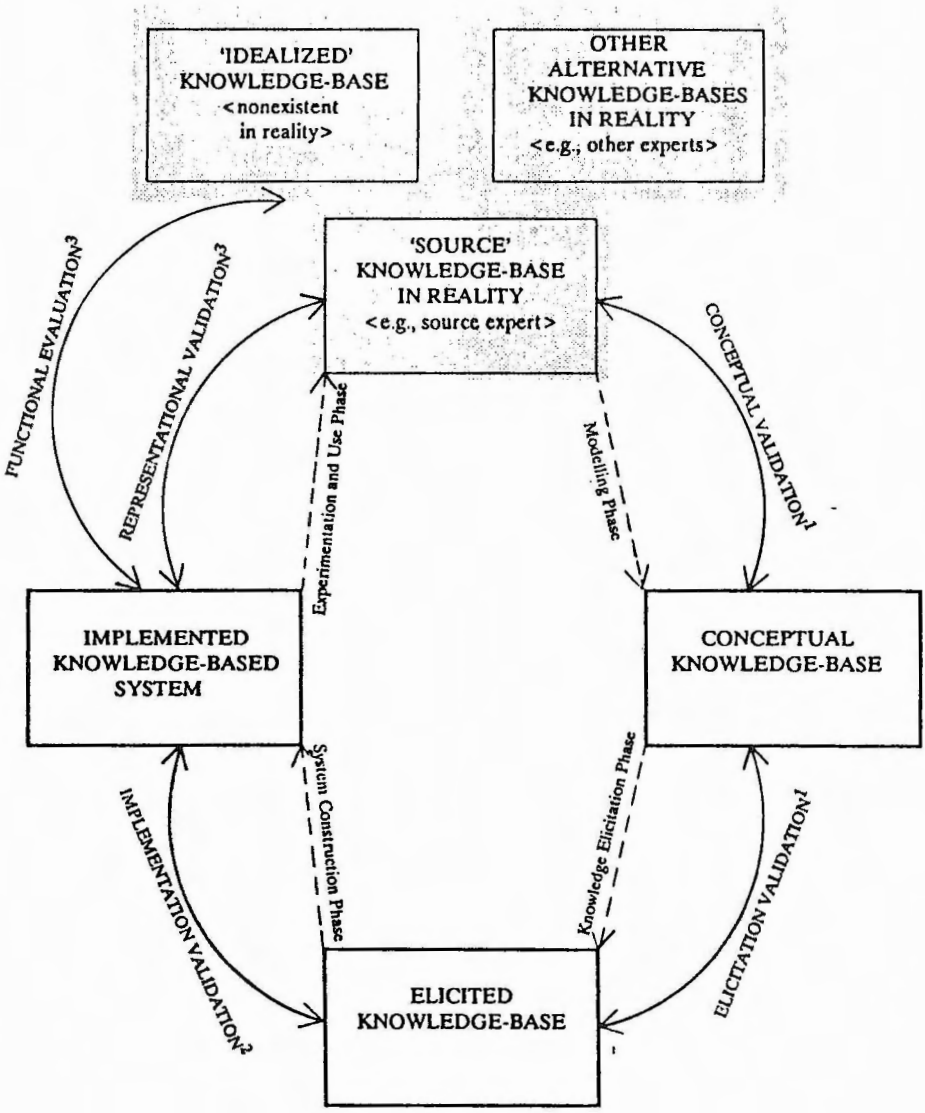
Vandierendonck, A. (1975). Inferential simulation: Hypothesis-testing by Computer Simulation, *Nederlands Tijdschrift voor de Psychologie*, Vol. 30, pp. 677-700.

Weitzel, J.R., and Kershberg, L. (1989). Developing Knowledge-Based Systems: Reorganizing the System Development Life Cycle, *Communications of the ACM*, Vol. 32(4) April, pp. 482-488.

Woodward, B. (1988). Knowledge Acquisition at the Front-End : Defining the Domain, *Proceedings of the 3rd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November,1988, pp. 37.1-37.25.

Xie, S., Dumaresq, D.F., and Winne, P.H. (1988). PRED: An Interface Development Tool for Editing Primitives in Knowledge Representation Languages, *Proceedings of the 3rd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November, 1988, pp. 38.1-38.17.

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



Notes: [1]Conceptual and elicitation validation constitute Knowledge-Acquisition Validation.
[2]Implementation validation can be termed Verification.
[3]Both functional evaluation and representational validation constitute Knowledge-Based System Validation.

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

Assessment Of:	Type of Test	Definition	Examples of Validation Methods and Procedures
CONCEPTUAL KNOWLEDGE BASE	Conceptual KB Qualification	Evaluation of a KB with respect to the problem being solved (KB relevance and adequacy).	• By identifying major domain and expertise contents, structures and boundaries, Cognosys (Woodward, 1988) facilitates conceptual KB qualification, i.e., if expertise/knowledge is sufficient and suitable for solving the problem prior to undertaking costly KA activity. • Structured walkthroughs by domain experts, knowledge engineers and users to isolate inaccuracies, ambiguities and omissions.
	Conceptual KB Validity	Validation of the KB with respect to the expert source being modelled	• Performed by KE in a series of clarification interviews with the domain expert using Cognosys to redesign & test domain models. (Woodward, 1988) • Structured walkthroughs by domain expert.
	Formal Checking	Evaluation of conceptual KB with respect to modelling formalism.	• Automatically enforced in Cognosys (Woodward, 1988) in terms of integrity checks and proper representation formats. • Visual checking by KE.
TEST CASES	Test-Case Applicability	Evaluation of test-cases with respect to problem being solved.	• Besides the domain expert, potential users of the KBS, other experts, and other related parties should also be involved to determine applicability of test cases to the problem domain. This involves seeking evidence of the applicability of the cases to the intended problem.
	Test-Case Validity	Validation of test-cases with respect to the expert source.	• Ruberg et al. (1988) used a domain expert to validate (termed 'rule authoring') rules induced by an induction system from a set of about 500 test-cases.

TABLE 1: VALIDATING THE CONCEPTUAL KNOWLEDGE-BASE

Assessment of:	Type of Test	Definition	Examples of Validation Methods and Procedures
STRUCTURE	Structural Validity	Validation of the structure with respect to the *perceived* structure of source expert.	• Check cognitive primitives against those in conceptual KB model developed earlier using Cognosys (Woodward, 1988). • Shaw & Woodward (1988) asked source experts rate their own entailments as generated by KSS0. • Use multi-methods of KA from same expert (e.g., protocol analysis and KSS0) to increase reliability/validity.
	Structural Comparison	Evaluation of the structure of a KB with respect to structure of some other KB.	• Shaw & Woodward (1988) have experts evaluate entailments generated by other experts. • Shaw & Gaines (1988) use notions of consensus, conflict, correspondence & contrasts.
BEHAVIOR	Behavioral Validity	Validation of KB behavior (output) and expert behavior generated under same conditions.	• Manual sensitivity analysis and logical walkthroughs by source expert. • Johnson et al. (1988) ask expert to comment on the behavior of a given chunk of the KB. • Use contingency tables to assess completeness (O'Leary, 1988).
	Behavioral Comparison	Comparison of behavior of elicited KB with behavior other KB's under similar conditions.	• Manual sensitivity analysis and logical walkthroughs by other experts. • Johnson et al. (1988) ask other experts either through interviews or by using formal questionnaires to comment upon validity of KB.
VERIFICATION	Elicited KB Verification	Evaluation of Elicited KB with respect to conceptual KB specification.	• Verify against domain models created using Cognosys at conceptual KB development stage (Woodward, 1988). These would provide a standard for comparison at later stages (e.g. for logical consistency).
	Formal Checking	Evaluation of Elicited KB specification in terms of the elicitation formalism used (consistency and completeness checks).	• Johnson et al. (1988) propose mediating representations between verbal interviewing data and implementation oriented knowledge representation schemes. Their Systematic Grammar Network (SGN) provides a methodology to prepare an explicit description of the knowledge prior to implementation.
METHOD	Data Collection Veracity	Evaluation of the validity and reliability of test data collected from the source expert.	• Xie et al. (1988) argue that the interface of a KA tool determines the accuracy/efficiency of a domain expert's description of knowledge, and the system's representation of that knowledge. Thus, the KA tool interface impacts data collection veracity and validation.

TABLE 2: VALIDATING THE ELICITED KNOWLEDGE-BASE

Assessment of:	Type of Test	Definition	Examples of Validation Methods and Procedures
STRUCTURE	Structural Validity	Validation of KBS structure against *perceived* structure of real-system.	• Enand et al. (1988) note that if the KBS contains any extra links that a human expert would not normally go through, validity is hindered.
	Structural Comparison	Evaluation of KBS structure against structure of other KBS.	• Similar to Enand et. al (1988)'s "Domain Validation". Testing under controlled conditions and third-party expert evaluations in domains too difficult or expensive to induce.
BEHAVIOR	Behavioral Validity	Validation of KBS behavior with respect to source expert.	• Turing-test comparing the behaviors of KBS and the source expert. • Source expert evaluates KBS output.
	Behavioral Comparison	Evaluation of the KBS behavior against non-source experts.	• Get other experts to rate recommendations produced by source expert. • Compare KBS output with known test-cases. • Compare KBS output to real system output. • Compare against predictions of theory.
	Sensitivity of Behavior	Comparison of KBS behavior (output) under different scenarios.	• Relates to Procedural Optimization of Enand et al.(1988): how adequately the KBS achieves the desired hit-rates, false-positives, & false-negatives. • Parallel testing • Regression testing
VERIFICATION	Implementation Verification	Evaluation of KBS with respect to the elicited KB specifications.	• Ruberg et al. (1988) describe the CIRCE verification system used to check the logical consistency of a rule-base after each major modification at the implementation level. • Evaluate redundancy, conflict, circularity and incompleteness of knowledge-base rules.
	Formal Checking	Evaluation of KBS in terms of structures & rules of the implementation formalism.	• Code walkthroughs, syntax checking, rule execution tabulation, code testing and debugging.
	Software Quality Assurance	Evaluation at KBS with respect to software quality standards.	• Performed by having implementation specialists evaluate the design quality of the KBS. Check for modularity, loose coupling, tight cohesion, and proper documentation.
USABILITY	Usage Comparison	Validity of KBS use against expert use.	• Measures of KBS utilization compared to retrospective records of expert utilization.
	Interface Validity	Validation of KBS interface characteristics with respect to the sources expert.	• Data is collected from interface comparisons of the KBS and the source expert, and by measuring users' perceptions. • Independent evaluations of interface by third-parties (Ribar, 1988)
METHOD	Data Collection Veracity	Evaluation of the validity and reliability of test data collected from expert.	• Shaw & Woodward (1988) contrast the entities and attributes given by experts at two points in to the data collection veracity.

TABLE 3: VALIDATING THE IMPLEMENTED KNOWLEDGE-BASED SYSTEM

I. CONTENT-RELATED VALIDITY EVIDENCE

Definition: evidence to assess the adequacy of the knowledge acquisition process with which the specified domain has been sampled and to determine the adequacy of the knowledge generated.

Assess the appropriateness, relevance and adequacy of the expertise sources selected and the test-cases to be used. Demonstrate that the content elicited is representative of the domain and excludes items outside its boundaries.

Evidence could be collected through structured walkthroughs, use of hypothetical scenarios and actual test-cases, assumptions challenging, and prospective user evaluation.

II. CRITERION-RELATED VALIDITY EVIDENCE

Definition: evidence relating to the assessment of a KBS by demonstrating a relationship between its behavior and some other criterion measure.

A. *Behavior (output)*
Evidence collected by comparing KBS output (through the Turing test procedure) to *source expert's* behavior to evaluate correspondence, at one or more points in time (intra-expert tests).

B. *Structure*
Comparison of evidence about process collected through: a) process tracing (reasoning process of functional KBS), and b) protocol analysis (collected from *source expert*) in order to evaluate correspondence in terms of:

- domain, construct, and range correspondence
- search-scope correspondence
- cognitive-primitives and meta-knowledge correspondence
- operational and event correspondence

III. CONSTRUCT-RELATED VALIDITY EVIDENCE

Definition: evidence relating to the assessment of whether a KBS (measure) is an appropriate operational definition of the source of expertise (construct) it purports to measure.

Developing an operational definition of the construct (expertise) and developing hypotheses about how it relates to other measurable variables. Seek *convergent and divergent evidence* by comparing KBS structure and behavior to:

- other experts (inter-expert tests)
- predictions of theories
- established facts in domain
- results of other KBS or simulation models
- known test-cases collected previously
- actual consequences of the output in reality

Evidence could be collected by means of Turing tests, process tracing, protocol analysis, walkthroughs, correlation tests, grid exchange analysis (using KSS0), etc.

TABLE 4: TYPES OF VALIDITY EVIDENCE

A philosophical basis for knowledge acquisition

P. Compton and R. Jansen†

Garvan Institute of Medical Research, St. Vincent's Hospital
Sydney 2010, Australia
†CSIRO Division of Information Technology
PO Box 1599 Macquarie Centre
Sydney 2113, Australia

Abstract

Knowledge acqusition for expert systems is a purely practical problem to be solved by experiment, independent of philosophy. However the experiments one chooses to conduct will be influenced by one's implict or explicit philosphy of knowledge, particularly if this philosophy is taken as axiomatic rather than as an hypothesis. We argue that practical experience of knowledge engineering, particularly in the long term maintenance of expert systems, suggests that knowledge does not necessarily have a rigorous structure built up from primitive concepts and their relationships. The knowledge engineer finds that the expert's knowledge is not so much recalled, but to a greater or lesser degree 'made up' by the expert as the occasion demands. The knowledge the expert provides varies with the context and gets its validity from its abiltity to explain data and justify the expert's judgement *in the context.* We argue that the physical symbol hypothesis with its implication that some underlying knowledge structure can be found is a misleading philosophical underpinning for knowledge acquisition and representation. We suggest that the 'insight' hypothesis of Lonergan better explains the flexibility and relativity of knowledge that the knolwedge engineer experiences and may provide a more suitable philosophical environment for developing knowledge acquisition and representation tools. We outline the features desirable in tools based on this philosophy and the progress we have made towards developing such tools.

Introduction

We wish to consider the philosophical question of the truth value of knowledge. This is not a question regarding the truth of a particular piece of knowledge, or a question regarding the ability of an expert, but a question about knowledge in general. In what way is knowledge really a representation of reality, how good a representation is it, how good a representation can it be. These are obviously time honoured questions in philosophy and we do not presume to suggest new answers but we do suggest that a consideration of epistemology may be important for developing appropriate knowledge acquisition tools, for obviously if you are trying to acquire something, your theory on what you are acquiring will influence your methods. Nor do we suggest that the problems we will identify are unknown to the artificial intelligence community, in fact there are many many practical attempts to deal with various aspects of these problems. We suggest however a better epistemology may allow better solutions to be developed.

There has been some consideration of epistemology in recent years by workers such as the Dreyfus brothers (1988) who attack the whole artificial intelligence (AI) venture, or others such as Winograd and Flores (1987) who suggest that AI's real achievements will be somewhat different from the achievement of machine intelligence. The line of the Dreyfus attack is that since artificial intelligence makes incorrect assumptions about the nature of knowledge it cannot succeed. One response of the AI community is that since AI does work and we can build expert systems, then the philosophy must be correct. Our aim here is to avoid this polarisation by focussing on a

replacement or modification rather than a rejection of what we perceive as the current AI philosophy of knowledge.

The prevailing epistemology is the physical symbol hypothesis of Newell and Simon (1981). Essentially this hypothesis considers that knowledge consists of symbols of reality and relationships between these symbols and that intelligence is the appropriate logical manipulation of the symbols and their relations. Although Newell an Simon developed these ideas in the 50's as a basis for AI, they have a long and time honoured philosophical ancestry, through the early Wittgenstein, back through Descartes to Plato with his archetypes.

AI research then is based on the assumption that one should be able to obtain in some way, fundamental atoms of knowledge and the logical relations between these atoms, and from these reassemble knowledge; it has an essentially reductionist strategy. The extension of the physical symbol hypothesis is the "knowledge principle", that is, the success of an expert system does not depend on the sophistication of its inferencing or reasoning strategy, but on the amount of information it contains on how symbols are interrelated, that is the amount of knowledge it contains. (Feigenbaum 1977). The fullest expression of this is the "knowledge is all there is" hypothesis (Lenat and Feigenbaum 1988), that is, there are no control or inferencing strategies that cannot themselves be contained in knowledge. Lenat (1988) would suggest that the CYC project, aimed at capturing much of common sense, or consensus reality, in a knowledge base is proceeding faster and faster because a fair number of the primitives out of which more complex knowledge is built have now been captured. Lenat would imply that the increasing success of the CYC project is a verification of the underlying philosophy.

Because the physical symbol hypothesis works so well it has become not one philosophy amongst others but an absolute viewpoint for AI workers. Thus the conventional explanation of the difficulty in acquiring knowledge is that experts don't communicate the underlying knowledge very well; e.g. as Waterman puts it ". . . . . the pieces of basic knowledge are assumed and combined so quickly that it is difficult for him *(the expert)* to describe the process." (Waterman 1986). With this type of perception, the problems in knowledge acquisition become, and have been over the last years, How do we really get to the bottom of the expert's knowledge, how do we really get down to the basic bits and their relationships, which can then be rebuilt into a powerful knowledge base? Many sophisticated techniques (eg AQUINAS, Boose and Bradsaw 1987) have been developed and some of these are available commercially (NEXTRA from Neuron Data (Rappaport and Gaines 1988)) which facilitate getting to the "bottom" of knowledge, and these have contributed and will contribute greatly to expert system development.

However there are clearly some problems. Shaw (1988) notes that contrary to the expectation of a knowledge engineer, different experts not only talk about a common topic of expertise in quite different terms, but are quite happy to disagree on terminology, without necessarily making this explicit. They use the same terminology in conceptually different ways and would appear to have different knowledge structures. Although these findings can of course be made to fit with the physical symbol hypothesis, they at least raise the question of whether there may be a better philosophy of knowledge than one whereby knowledge is made up from some sort of absolute primitive elements. Wittgenstein, in his later years, gave up logical atomism precisely because of this sort of failure to find the primitive concepts on which knowledge is built. (Wittgenstein 1953).

A different type of question is, What happened to all the prototype expert system? Clearly there have been far more systems prototyped than deployed? The phase of prototyping an expert system is the hopeful phase. You have got 90% of the knowledge into the system, and you are confident that if only you persevere, you will overcome the problems and the system's knowledge will become complete. Generally there are no further reports beyond the prototype stage, but where there are, a different story emerges. McNamara et al (1988) have started again with a new design to build an expert system which they were confident was close to completion in 1987 (Lock Lee et al 1987). After four years in routine use, knowledge addition to XCON/R1 still involved 4 full time knowledge engineers, although this was at least partly because the domain itself was expanding with new computer hardware to be configured (Bachant and MacDermott 1984). More recently a reimplementation of XCON has been commenced in an attempt to make it more maintainable (Soloway et al 1987). GARVAN-ES1 has doubled in size since 1984 when it was put into routine use although the knowledge domain has not expanded and the system was 96% correct when deployed (Horn et al. 1985 and Compton et al. 1988). We will argue below that

such problems are not the brittleness, or ultimate stupidity, of expert systems due to lack of common sense that Lenat would fix with CYC, but problems related to what knowledge is.

We can also note that the apparent increased deployment of expert systems at present is somewhat misleading as an indicator of improvement in expert system building. There seems to be a fairly major shift in the type of expert systems being built. In the past medical diagnosis was the expert system paradigm, whereas now the focus is on areas such as finance and process control. These are quite different types of problems with quite different requirements of expertise. In medicine the starting point is that each individual patient should have the very best care and diagnosis available. In practice we then step back from this goal because of costs, human frailty, organisational problems and so on. An expert system in this environment must be 'truly expert', that is it must make no mistakes that could be picked up by a human. Schwartz et al (1987) on behalf of the medical AI community have given up this goal until large advances in technology are made. By contrast, in areas such as finance it is sufficient that the errors an expert system (or an expert) makes are outweighed by its successes, according to the relevant profit criterion. Any improvement above this minimum is pure profit.

We would suggest that material for philosophical reflection is less obvious in these "error tolerant" systems than in systems where total expertise is demanded. We will consider here one attempt to perfect an expert system.

The maintenance experience

GARVAN-ES1 is a small medical expert system which consists of 96 rules which provide simple reduction of numeric and string data to classes and 650 production rules (without disjunctions) comprising the knowledge (Horn et al 1985). Its knowledge is entirely heuristic and contains no deep knowledge of the domain. The system is used to provide clinical interpretations of data for reports from a laboratory which measures thyroid hormone levels. It is forward chaining only as all the data is available prior to running the expert system during report generation. The system is a heuristic classification system (Clancey 1985) and chooses from between 60 different interpretations and provides about 6,000 interpretations per year. The system has been in routine use since mid 1984 and was identified by Buchanan (1986) as one of the first four medical expert systems to reach routine use. Most importantly it has been continuously maintained since its introduction (Compton et al 1988). Maintenance is made possible because each report issued by the system is checked and signed by a human expert or referred to the knowledge engineer if the clinical interpretation is unsatisfactory. Although 96% percent of the reports were acceptable to experts when the system was introduced into routine use, the knowledge base has more than doubled in size over the four years (Fig 1). with acceptance of the reports now apparently 99.7% (see Compton et al 1988a,b for the significance of this figure).

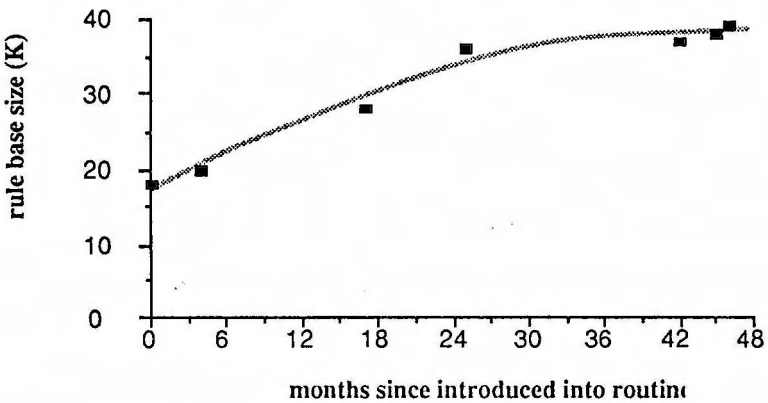


Fig 1. The growth of GARVAN-ES1 since its introduction into routine use in mid 1984. The Yaxis gives the number of characters in the rule base, and is best used to indicate the relative change in size of the knowledge base. The data points represent some copies of the rule base that were kept. Maintenance was carried out more regularly than these data points suggest. The curve was drawn by eye for illustrative purposes.

Compton et al (1988) provide an example of an individual rule that has increased four fold in size during maintenance and there are many examples of rules splitting into three or four different rules as the systems knowledge was refined. There are also examples, albeit very few, of later experts deciding that the knowledge contained in some early rules was completely spurious.

The most important resource for knowledge maintenance is a data base of "cornerstone cases". These are cases which at some stage have required a change in the system's knowledge. Every time the system's knowledge is changed the interpretations for all the cornerstone cases are checked to see that the additions to the system's knowledge have been incremental and have not corrupted the knowledge.

A basic experience of maintaining GARVAN-ES1, (and all knowledge engineering?) is that the knowledge one acquires from the expert has to be extensively manipulated so that it can be added to the knowledge base without corrupting the knowledge already contained. The difficulty of doing this is exacerbated during the maintenance phase not only because the knowledge base is becoming more complex, but because the expert and knowledge engineer are no longer closely familiar with the knowledge communicated during the prototype phase; in fact different experts and knowledge engineers may be involved, as has happened with GARVAN-ES1. Experts will still unfailingly give interpretations and good reasons for the interpretations for every set of data presented to them, but they are no longer at all familiar with the knowledge structure to which this knowledge is to be added. Shaw illustrates that experts have different knowledge structures concerning the same domain. (Shaw 88). We will argue below that even the knowledge provided by a single expert changes as the context in which this knowledge is required changes.

Figure 2 summarises the maintenance experience. The expert provides what appear to be very simple clear rules, however when these are added to the system they subsume and conflict with prexisting rules and must be extensively manipulated before they can be added to the knowledge base. This is not just a problem because the structure of a knowledge base is influenced by the inference strategy used on it (Bylander and Chandrasekaran 1986); the problem is in the knowledge provided by the experts. Further, the problem cannot be resolved by tools which check knowledge bases for subsumption etc (Suwa et al 1984) because rules which have no logical overlap may still be satisfied by a single data profile.

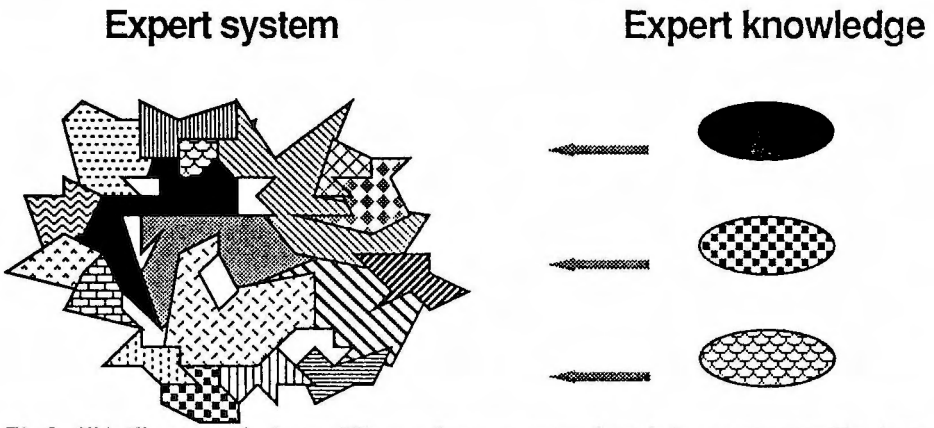


Fig 2. This illustrates the large difference between expert knowledge as expressed by the expert and as contained in an expert system. The diagram suggests how rules as expressed by experts must be manipulated and changed to ensure that they do not overlap with other rules. The physical symbol hypothesis would hold that the structure on the left is some sort of a map of knowledge of a domain.

Our previous study (Compton et al 1988) provides a detailed example of maintenance. This example illustrated that in the maintenance phase the context influences the rules that the expert provides, a very non controversial proposition. For example if a set of data was given interpretation B by the expert system when it should have been given interpretation A, then the

rules the expert provides to correctly reach interpretation A will differ from those when the expert system had given interpretation C rather than A. No one would dispute that this will occur, that the context in which questions are asked will influence the answers. However the physical symbol hypothesis in fact suggests that something different should occur, that we should be able to obtain from experts the primitives and structure of their knowledge, that we should be able to obtain a rule to reach conclusion A instead of a group of more relative rules which conclude A rather than B and A rather than C etc. The physical symbol hypothesis suggests that there should be underlying rules that provide a more absolute foundation for the knowledge, rules to reach A rather than not A, i.e. context free rules. It has been harder than expected to find such foundations, hence the new knowledge acquisition tools (NEXTRA etc) aimed precisely at broadening the context to enable the expert to see all the bits and pieces of knowledge they provide, so that the structure can be found. These work, structures can be found, but they appear to differ from expert to expert (Shaw 1988).

The experience with GARVAN-ES1 suggests, rather than arriving at the ultimate structure, the knowledge acquisition process goes on and on, albeit more slowly; even the rule changes introduced during the maintenance example alluded to above and described previously (Compton et al 1988) have required further changes introduced during the writing of this manuscript. The rule base has doubled in size in trying to go from an accuracy of 96% to an accuracy of 99.7% (probably not as high as this in fact(Compton et al 88). When one examines the composition of GARVAN-ES1's data base of cornerstone cases, as a record of the growth of the system, one finds that rules for even the simplest and most clear cut interpretations are not fully correct when they were first entered, and it was necessary for the rules to misinterpret cases and fail to interpret cases a many times before they became reliable (Compton et al 1988). Experts however have little trouble in interpreting these cases and providing new rules; which they will then endlessly change when provided with other cases. It should be noted that the domain of interpretation of thyroid reports is a simple domain in medicine where you would expect it to be easy to get complete knowledge.

A philosophy of knowledge

This maintenance experience suggests that knowledge is not sometimes, but always given in context and so can only be relied on to be true in that context. The best philosophical foundation for this seems to be the philosophy of Karl Popper (Popper 1963). Popper suggests that we can never prove a hypothesis, we can only disprove hypotheses. Obviously we cannot disprove all alternate hypotheses, we may not even be able to formulate all the alternatives. What we can only do is to disprove the limited number of *likely* alternatives. We suggest that this explains the phenomenon in knowledge aquisition that knowledge, the rules for reaching some conclusion, always seem to be context dependant. What the expert is doing is identifying features in the data that enable one to conclude that a certain interpretation *is preferable to the small set of other likely interpretations*, and of course the likely alternatives depend on the context. This is quite different from reporting on how one reached a given conclusion. These are not novel suggestions. It is well established in medicine for example, that clinicians use hypothetico deductive methods, that they attempt to distinguish between a small set of hypotheses (Griner 81). In earlier studies we have suggested that it is precisely the hypothetico deductive approach which allows clinicians to deal with the errors that occur in laboratory data. (Compton et al 86)

We would suggest that the knowledge that experts provide is essentially a justification of why they are right, not the reasons they reached this right conclusion. This phenomenon is familiar also in arguments. Arguments are always about showing how we are correct in terms that will convince the other person. No one is ever wrong but the context in which one is right changes during the argument! Similarly when an expert is explaining something, his explanation varies depending on whether he is explaining it to a layperson, a novice or a peer. If the expert was explaining, giving the "real" reasons, rather than justifying, they would provide the most detail to the layperson, who knows nothing about the area and the least to a peer, who knows a lot about the area. In fact the reverse happens, the layperson gets the most superficial explanation, because generally it requires very little to justify that the expert's hypothesis is better than the naive hypotheses of the lay person, while the fellow expert, precisely because he is an expert, will require more rigor in the justification. It has been easy to witness this phenomenon with GARVAN-ES1. If the knowledge engineer takes a difficult case to two experts independently, he will get two fairly simple, but sometimes slightly different rules, although each will apparently peform equally well in the expert system. If the knowledge engineer then brings the experts together and asks which rule is right, a

very complex discussion is liable to ensue, as the experts (politely) attempt to prove to each other that their rule is better, normally resolving the question, by agreeing that their rules apply in different contexts and are complementary..

It seems to us that the proposition that the knowledge we communicate is a justification in a context, not how we reached our conclusion, is universally true of all knowledge. Popper, although his ideas are mainly applied to scientific knowledge, applied them to all knowledge (Popper 1963). A baby makes hypotheses about the world, which are then tested against experience and new hypotheses constructed, or the old ones modified, if the hypotheses are seen to fail. This would seem to fit closely with Piaget's observations of development in early childhood. (Piaget 1929). There have of course been additions to Popper's falsification hypothesis. Kuhn has drawn attention to paradigm shifts in science, where investigation shifts to new types of hypotheses for other reasons than the falsification of earlier hypotheses.(Kuhn 1962) Lakatos (1978) points out how falsification of hypotheses may result in riders being added to allow the major hypothesis to continue, although it becomes very unwieldy. An example would be Ptolemy's addition of epicycles, when it was found that planetary orbits were not circular, in order to maintain the primacy of circular motion. These developments do not disagree with the essential proposition of interest here, that the knowledge we communicate to each other is a justification of a judgement expressed in a specific context. They will be considered briefly later.

This viewpoint has a number of consequences. If all knowledge is only true in a context, then all knowledge is relative, it only exists in relation to other knowledge, there is no absolute underlying knowledge on which the rest of knowledge is built. We can note that AI does attempt to deal with the problems that flow from this. For example circumscription (McCarthy 1980) is an attempt to deal with the exceptions to commonsense beliefs, that always occur, but they are still seen as only exceptions, rather than as indicators of the fundamental relativity of all knowledge.

How then can knowledge be in any sense true, our original question? Before attempting to answer this we must consider the most common response to such suggestions; that is, that underlying primitive knowledge is made up from sense data. On the contrary, it is clear that what we perceive depends on the context in which the perception takes place. There are well known examples such as when one wears glasses which invert the view seen. After a while the inversion is corrected and the world is seen the right way up (Stratton 1897). A small model of a distorted trapezoidal shaped room can be built, but with viewing apertures with the appropriate prisms so that the room appears of appropriate proportions when viewed through the prisms; the model table, windows doors etc all have their distortion corrected. If one is asked to touch various parts of the model with a stick poked in from above, it is difficult because one has a misleading view of the room. If one perseveres and learns to touch the different objects easily, the true distorted shape of the room is *seen* although one is still looking through the correcting prism.(Ames 1951, described by Bateson 1979) Clearly one's expectation determines what one sees, i.e one's intellectual context determines the so-called prime data of sense. "The Logic of Perception" by Lock (1980) is entirely devoted to demonstrating in a formal way that the mind determines what one is able to perceive. Colour perception provides another interesting example of the same phenomenon. There are a number of tribal groups throughout the world who perceive fewer colours than Western man (Treitel 1984). This is not due to any impairment of visual apparatus; Taylor (1984) records how as a chemistry teacher in Nigeria, he found it very difficult to teach the use of litmus paper, because students, saw red and blue litmus paper as the same colours. However after much practise, they were able to distinguish the two, that is they were able to see red and blue as different colours. McLuhan and Parker(1969) note that perspective, the vanishing viewpoint, was introduced into art at a specific stage of history. Art prior to this time showed great technical skill but no perspective. There was then a sudden change to use of perspective. Since the artists surely had the ability to paint perspective, but didn't, perhaps they didn't *see* things in perspective till perspective was discovered or created. In fact, McLuhan and Parker's comments are much stronger and they would argue that there has been a continuing change in our perception of space and time.

Sense data then cannot be viewed as providing the primitives out of which other knowledge is built, because the sense data available to us is determined by our mind, by the knowledge we have and the context that this provides for perception. It is possible to argue that the chemical changes resulting from perception may be accumulated and recorded below consciousness. Such data may exist but is clearly irrelevant if we are seeking for the expert's knowledge, that is what the expert is

conscious of and can reason about. We can also note that an appeal to chemical storage of information as providing the primitives of knowledge is contradictory in that it is based on ideas of chemical storage, which is itself part of knowledge.

Although we have not used their arguments to support our position we can also note the similarity of this viewpoint to some of modern physics, where the world that is discovered through experiment, is determined by the knowledge we already have. The concrete realness of space and time that common sense suggests, are our creation to express our experience, rather than some absolute of reality in itself. (d'Espagnat 1983 for a restrained exposition of these ideas).

The picture of knowledge that has emerged then is one where knowledge only has meaning in relation to other knowledge. Any set of apparent primitives that one may find only have their meaning in relation to other knowledge. It is interesting to note that the new knowledge elicitation techniques, based on psychological rather than philosphical theory, treat knowledge in this way and attempt to explore knowledge in terms of relationships. But we are not back to a version of the physical symbol hypothesis whereby the set of relationships of the whole knowledge structure provide the foundation for knowledge. On the contrary our argument has been that any knowledge structure that is elicited is itself always provided in a context and cannot be guaranteed not to change when the context changes, and we note again Shaw's observation of the different knowledge structures experts have of the same reality. (Shaw 88)

What then is knowledge, how can we communicate knowledge if it is all relative, what can it mean to judge that something is true? The most satisfactory theory seems to be that of Lonergan (1959) who in some 900 pages describes the concept of insight, and its role in the different kinds of knowledge. Essentially the act of insight is our act of recognition that something does make sense. The physical symbol hypothesis viewpoint sees only knowledge. Insight is the act whereby we perceive that this knowledge makes sense of reality, expresses some intelligibility in reality. Lonergan specifically chose the word 'insight' to express, the excitement, the flash of discovery. Archimedes "Eureka!" was because he discovered a theory *that made sense of reality*. Once the theory was expressed, one could examine its logical structure, and express it in terms of a physical symbols, relations and logic, but first of all a theory had to be created and the soundness of its semantics as well as its logic recognised in the act of insight. Some of the earlier discussion may have seemed to align us with Kant, and with his more idealist followers, that knowledge and sensation are something internal with a tenuous or non-existent connection to reality. Lonergan's approach is descended from the realism of the Scholastics, a realism divorced from the essentially naive realism of rationalism. We are directly and ontologically in touch with reality in insight, but our expression of this is intrinsically different from reality; knowledge and perception never contain what is known. We never fully grasp and express and contain in our knowledge the intelligibility of reality, which of course leads back to Popper's hypothesis. Knowledge never expresses reality but we progress in knowledge, through disproving hypotheses in the unending search to express insight truly. We use our knowledge to try and express insight, but it is also part of the insight process, because the building of the knowledge is part of the process of seeing the intelligibility in reality.

Communication of knowledge can then have a two fold function. We can use it as a justification, to show that out of the alternative hypotheses, ours is the best. Or we can use it to try and convey insight, to enable the other person to recognise, even partially and differently, the intelligibility we have seen. As we know the worst kind of conversations are where people are trying to show each other they are right and the best kind are where they are genuinely trying to see and enable each other to see their respective insights. Insight can of course often be expressed very badly and rigorous debate, scientific or otherwise, is necessary to see whether the hypothesis used to express the insight is wrong on some other ground. This does not mean the insight is wrong, but the expression of the insight is contradicted by the knowledge which expresses other insights. Even if the way the new insight is expressed seems perfect, it is by definition an hypothesis to be eventually supplanted by a "less wrong" or perhaps more interesting hypothesis.

We do then build up a body of knowledge, a knowledge structure. The trouble with this structure, is that it does not get its value because it is assembled from more primitive, more true, elements. It get its value because the various hypotheses that express the various insights don't conflict, they are consistent and coherent. But they are only consistent and coherent, as far down as they have been checked, and they have only been checked as far as demanded by the interaction of the

various hypotheses, and the insights they express. As these change and develop the knowledge structure changes. There is no doubt there is a knowledge structure, but it gets its validity from the consistency and coherence of the structure, not from the elements that make it up. The structure of course also gets its validity, from how well it expresses insight and retains contact with insight. If the expressed knowledge is taken as the truth, disconnected from insight, the body of knowledge rapidly becomes corrupt. The weird aberrations that arise in philosophies, religions, civilisations if they lose their grounding insights attest to this phenomenon.

We can also note that insight is not something that happens automatically and uniformly. An expert can probably be defined as someone whose technical knowledge cannot be faulted and whose decisions will meet with the approval of his peer group and those who would seek his opinion. But there are also 'experts' in every discipline who are lateral thinkers, who are able to propose solutions for completely novel situations. At the other end of the scale are those who have no insight and attempt to apply textbook reasoning with disasterous results. Lonergan treats in detail the factors involved in insight.

Implications for expert systems and knowledge acquisition.

Any system that attempts to reach the fundamentals of knowledge on which all the other knowledge can be built even in a specific area , will not work, or will work by good fortune rather than design. The design principle of the vast body of expert systems, that you should try to get the general rules, that knowledge should be separated from context and generalised, is philosophically unsound.

Experts will always give knowledge in context, with rules that conflict, subsume, overlap etc. If you push them, they can express a body of knowledge that is more complete, consistent and coherent. But you have to push them, they are not digging out some underlying knowledge structure, they are constructing something, making something up to satisfy the knowledge engineering demand. Hence Shaw's finding that the knowledge structures that experts made up differed; and she further notes that the experts were happy to live with these differences and didn't see it as a matter of concern.

It is not suprising that CYC is starting to assemble useful "primitives", and that analogy is an increasing powerful tool in its repetoire; we do all communicate, so we must have something in common. But our communication is based on being able constantly to revise the language we use to express insights. If CYC becomes a monolith, what will happen with paradigm shifts. Kuhn has noted how scientific hypotheses of interest change for other reasons than just falsification. So can common sense; new insights are always possible, and can change the whole emphasis on what are the important areas to be made consistent and coherent in knowledge. The function of the artist (and still the scientist?), is to open up new insights, to find new ways of viewing reality.

We would suggest then, that it is fundamental to the expert system enterprise, that knowledge NOT be generalised when it is acquired. It is fundamental to try and also record the context in which the knowledge is acquired, for it is in the context that it still has its closest link with insight, and therefore the most validity. The knowledge can always be generalised later. Inductive learning methods such as ID3 (Quinlan et al 1987) provide examples of generalising from the highly context based knowledge of individual cases. We are proposing here that even where knowledge is elicited from experts, it should be recorded in context. One attempt at doing this is described in Appendix A and Compton et al 1988. Context of course cannot be fully recorded for we cannot know everything that makes up the context, precisely because it is the context.

The second point is that since the knowledge base will have to change it must be designed for change. One should be able to investigate and interrogate the knowledge from any viewpoint that is desired and change any aspect of the knowledge without corrupting the whole, or at least be able to control the changes fully. Appendix 2 and Jansen and Compton 1988a & b and 1989a & b *(these proceedings)* , describe our progress towards this goal. The essence of our approach is to use data base technology, so that knowledge, eg rules, can be broken down into component parts and stored in relational form, with all the facilities for maintenance that this implies. In one sense the goal is maintenance, but maintenance is not an extra, a problem that has to be dealt with but is just an instance of the fundamentally fluid nature of knowledge.

The underlying problem is that computers are machines that manipulate symbols. The physical symbol hypothesis is perfect for describing computer intelligence. Computers demand a reductionist approach, where the whole, the body of knowledge is assembled from the parts. Unfortunately, human knowledge isn't made up of such primitives. It may look like it on some occasions, but essentially it is made up as occasion demands, and only within the context in which knowledge is provided, where insight is at work, can it really be guaranteed to have any consistency and coherence,.

This doesn't mean computers shouldn't be used for storing knowledge. On the contrary we suggest that they will be major tools for developing consistent knowledge bases. And perhaps one of their major roles in the area of knowledge and intelligence, will not be to provide expertise, but to enable humans to examine how a new expression of a new insight relates to the totality of what is "known". We see that perhaps the most powerful role for the computer in its present technological form is as a hypothesis tester rather than the dispenser of intelligence. We do not question, however that expert systems can and do work and have an important future and we hope that our own work is some contribution to this future. But we suggest again that whatever the final form and use of computer knowledge bases, we will make more progress if we don't assume that the knowledge that is going into the computer has come from a computer like thing.

Appendix 1 - Ripple down rules

We have proposed elsewhere (Compton and Jansen 1988) a simple structure aimed at capturing, at least partially, the context in which knowledge is obtained from an expert. Maintenance incidents with GARVAN-ES1 always occur because the expert system misinterprets or fails to interpret a case. The context in which a new rule is obtained from an expert always includes the information that there was no interpretation made or a specific interpretation was wrongly selected by the expert system. In the case of the wrong interpretation, we use as the context the specific rule that gave the wrong interpretation. So the new rule that the expert provides only becomes a candidate to fire if the rule that previously fired incorrectly is again satisfied. We do this by including a LAST_FIRED(rule number) in the premise of the new rule, which must be satisfied as well as the other conditions.

All rules in this system include this LAST_FIRED(rule number) condition, but rules which are not added as corrections have LAST_FIRED(0) indicating that they are the initial candidates to fire when the data is first presented to the system. The rules are only passed through once, from the oldest to the newest, and of course once one of the LAST_FIRED(0) rules has fired none of the other LAST_FIRED(0) rules is eligible as now a specific rule has fired. Each rule added is put at the end of the list of rules. When an error occurs because the system has failed to make an interpretation the new rule becomes a new LAST_FIRED(0) rule. The implicit context when such a rule is obtained from an expert and added is that none of the knowledge in the system covers this case. The rule obtained from the expert will almost certainly subsume or overlap other rules in the knowledge base - this is the standard experience of knowledge engineering. With ripple down rules however, the rule can be as general as the expert likes, because it will only be tested after all the other knowledge in the knowledge base has been considered, exactly the situation in which it was acquired from the expert. The structure of ripple down rules can be described in a number of ways. Fig 3 shows an alternative description of the knowledge structure.

This system means that rules can be entered into an expert system exactly as provided by the expert and the growth of the system's knowledge records the way in which knowledge was acquired from the expert. This is in complete contrast to conventional expert systems, where the aim is to generalise, and every rule (at the same level) has an equal opportunity to fire. However, the result of generalisation, is not clear general rules, but rules that become highly specific and opaque in trying to cope with the interrelations in knowledge in a general way.

In our previous study we described the incremental testing of, and addition to a set of ripple down rules to replace GARVAN-ES1, using a large data base of archived cases. This evaluation still is not complete but the essential behaviour of such rules is clear.

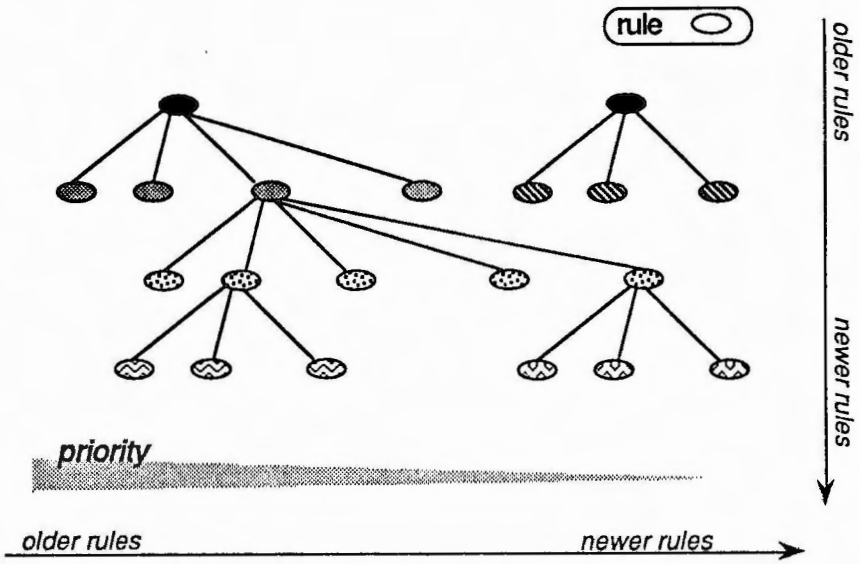


Fig 3. This is a simplified tree representation of ripple down rules. The two black elipses at the top of the tree are two LAST_FIRED(0) rules, so the precondition for either of these rules to fire is that no other rule has fired. The rule on the left, the earlier rule is considered first. If it doesn't fire, the next oldest LAST_FIRED(0) rule is considered. If a LAST_FIRED(0) rule does fire, then only the rules connected to it on the next level are candidates to fire. Again these are considered, one by one, left to right, the oldest to the newest added. Each of these has been added because the parent misinterpreted a case, but no further rule was able to fire. Each includes the LAST_FIRED(parent) condition. Once one of these daughter rules is added then the only the rules connected to it are candidates to fire.

The major feature of ripple down rules is that they can be added to a knowledge base far faster than conventional rules since the rules are added as is without modification. Secondly, since they are used only in context they have far less impact in corrupting the knowledge base. (We note again that this is not a problem that can be resolved with tools that check for subsumption automatically etc (Suwa et al 1984), because often the overlap between rules, is not in the rules themselves, but in the patient data profiles; the data from single patient may well satisfy a number of rules that don't overlap). In adding 534 rules to the ripple down rule knowledge base, in only 31 cases did an extra rule have to be added because the first rule added had changed the performance of the knowledge base. This is in complete contrast to conventional rule addition. A log of recent maintenance activity on GARVAN-ES1 showed that it had taken 31 attempts to introduce 12 changes, with of course extensive checking of the knowledge base along the way. Because new rules have little effect on the ripple down rule knowledge base they can be added far faster than with a conventional system. It is not to difficult to add 10 rules per hour in contrast to the often mentioned industry figure of 2 rules per day. Such rules do work and Fig 4 illustrates the accuracy achieved in the evaluation so far. This evaluation is not complete but it seems likely that the new rule base will be as accurate as the original without being much larger than its 650 rules. With 40 fold faster rule addition, even a doubling in size would be insignificant. It will be necessary to retest the knowledge base on a further large data base of cases since as there are now only about 1500 cases that have not been used as the basis for rule changes and which can legitimately be used as test cases.

Redundancy is obviously the major problem with this approach. Since knowledge is entered in context the same rule may end up being repeated in multiple contexts. Although redundancy has not been a major problem in the work so far it is still rather disatisfying for the knowledge engineer to have to add knowledge which he knows exists elsewhere in the knowledge base even if the addition is very easy. We are therefore investigating ways of generalising from the knowledge in this sort of knowledge base. We never throw away the knowledge in context, but we attempt to see if some generalised knowledge may be relevant on a second pass through the knowledge base.

In our first attempt at this we have implemented a LAST_DIAGNOSIS condition which essentially uses only the information implicit in the LAST_FIRED condition specifying the diagnosis or interpretation reached. This is currently under evaluation.

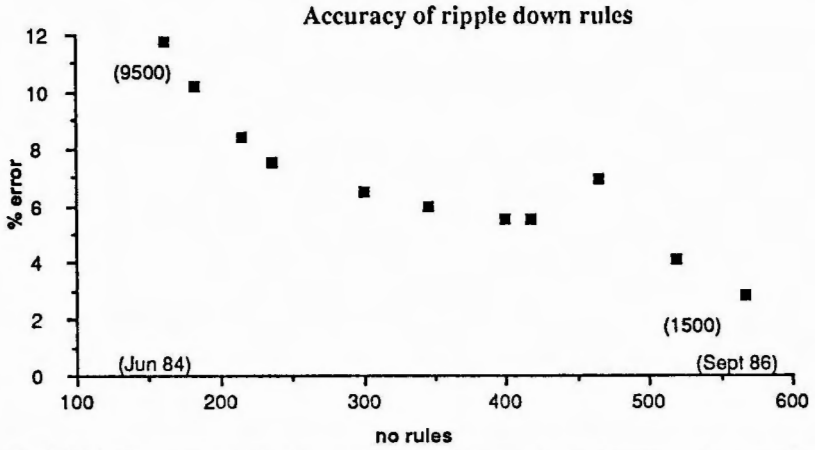


Fig 4. This figure illustrates the percentage of errors in the interpretations produced by ripple down rules when compared to the original interpretation produced by GARVAN-ES1, as a function of growth in the size of the rule base. The number of test cases reduces during the knowledge base growth from 9500 to 1500, as incorrect interpretations of test cases were used as the basis for further knowledge addition. Whether a case had been correctly interpreted or not, once it had been a even a candidate for rule changes, it was not reused to ensure the test was fully valid. The dates indicate the period the archived cases derived from.

Ripple down rules is just one very specific attempt at capturing knowledge in context and there are no doubt other and better strategies. But such strategies must be based on the recognition that knowledge has its primary validity in context where insight is active and that generalisation is a secondary activity. It is a mistake to throw away the specifics on the assumption that knowledge is built from generalised primitives.

Finally the reader may wonder why we did not use inductive learning techniques such as ID3 on such a large data base of cases. We have in fact used such techniques (Quinlan et al 1987), but inductive learning is not appropriate for maintenance where you are essentially dealing with the first apparent instance of an new data profile, and secondly our purpose here was to explore a new approach to knowledge acquisition and organisation.

Appendix 2- the knowledge dictionary

As well as capturing knowledge in context it is also essential that the knowledge in a knowledge base should be able to be examined in any context desired. We have noted the flexible way experts change the context in which they provide knowledge. Expert system knowledge bases per se do not allow this type of flexible access, so it is an area of major interest to develop techniques providing this type of better access to knowledge (NEXTRA etc) independent of the philosophical issues raised above.

Our approach has been to focus on the underlying representation, which may then provide a suitable basis for flexible access. One of the major issues is that the underlying representation should be normalised as far as possible. Knowledge bases have the same potential problem as large data bases. You do not want any of the knowledge to become segmented and isolated. If you have to examine or change some knowledge object, you want to be sure you have access to every occurrence and usage of that object and every relation it may have with other objects. Obviously the more the representation is normalised the more readily this can be achieved.

We have suggested (Jansen and Compton 1988a & b and 1989a & b-*these Proceedings*) that the conventional data dictionary (Dolk et al 1987) may be expanded to include knowledge and may thus provide a suitably normalised underlying representation as well as a link to conventional data bases. The use of an extended data dictionary which includes knowledge, that is a knowledge dictionary, then opens up the possibility of applying the results of software engineering research to knowledge design and implementation.

The allowed relationship model for the knowledge dictionary is described elsewhere (Jansen and Compton 1989a). Fig 5 provides an example of this sort of representation in that rules are broken up into their component parts and stored in a relational table. This allows relational calculus to be used to manipulate the knowledge base, with all the generality and power that this implies. The example in Fig 5 is of a production rule, but we have developed elsewhere ways of representing frames, semantic nets and the ripple down rules described above (Jansen and Compton 1989b-*these Proceedings*). The inclusion of other representations in the dictionary is the subject of further research.

PRODUCTION RULE

RULE(42)

IF FTI is high
and T3 is high
and TSH is undetectable
and not on_t4

THEN **thyrotoxic**

ELEMENT RELATIONSHIP TABLE

owner	relationship	member
-------	-------	-------
-------	-------	-------
RULE_42	presence	FTI_high
RULE_42	presence	T3_high
RULE_42	presence	TSH_undetect
RULE_42	absence	on_t4
RULE_42	outcome	thyrotoxic
-------	-------	-------
-------	-------	-------

Fig 5. The production rule (upper) can be represented as a set of tuples in a relational table, where the rule as an object has presence and absence relationships with various facts and an outcome relationship with an interpretation. This is a simplified representation as object types etc are not indicated.

Versions of the knowledge dictionary have been implemented in both Prolog and Hypercard on the Macintosh and RDB and Rally on the MicroVax, and the GARVAN-ES1 knowledge stored in relational form into the more fully developed Macintosh dictionary. An inferencing mechanism has been developed within the relational framework, although for many applications, a run time

knowledge base would be generated from the dictionary to be used with the expert system shell required.

Current work is directed towards developing the appropriate context strategies using pattern matching and table searching. For example we have previously described (Jansen and Compton 89a) a *why not* facility whereby one can query the reasons a rule did not match a certain data profile. This facility requires only simple data manipulation rather than rule parsing since the knowledge is stored as data. It is not hard to see how one may ask instead of, why not a rule, why not a specific interpretation, or certain class of interpretation? These are technical problems rather than research problems because the underlying representation supports the type of query required.

Acknowledgements
This paper is the result of a collaborative project between the Garvan Institute of Medical Research and the CSIRO Division of Information Technology. We wish to thank our colleagues in both institutions for their support, particularly Les Lazarus of the Garvan and Bob Colomb of the CSIRO. We also wish to thank Kim Horn and Ross Quinlan, without whom there wouldn't be a GARVAN-ES1 to base this study on, and the Garvan staff who have provided endocrine expertise over the years.

References

Ames, A Jr. "Visual perception and the rotating trapezoidal window" Psychological Monographs (1951); 65, whole no. 324.

Bachant, J. McDermott, J. "R1 revisited: four years in the trenches", The AI Magazine (Fall 1984); pp.21-32

Bateson, G (1979) *Mind and Nature: a necessary unity* Wildwood House, London

Boose, JH., and Bradshaw, JM. "Expertise transfer and complex problems: using AQUINAS as a knowledge acquisition workbench for knowledge-based systems." International journal of Man Machine studies (1987); 26 pp.3-28

Buchanan, B. "Expert systems: working systems and the research literature" Expert Systems (1986); 3(1) pp.32-51

Bylander, T. Chandrasekaran, B. "Generic tasks for knowledge-based reasoning: the 'right' level of abstraction for knowledge acquisition", Proceedings of the knowledge-based systems workshop, Banff, (1986); pp.7.0-7.13

Clancey. WJ, "Heuristic classification" Artificial Intelligence (1985); 27 pp.289-350

Compton, P. Horn, K. Quinlan, R. Lazarus, L. "Maintaining an expert system". Proceedings of the fourth Australian Conference on Applications of Expert Systems, (1988); pp.110-129

Compton, P. Jansen, R. "Knowledge in context: a strategy for expert system maintenance" Proceedings of the Australian Joint Artificial Intelligence Conference (AI'88) Adelaide (1988); pp283-297

Compton, PJ. Stuart, MC. Lazarus, L. "Error in laboratory reference limits as shown in a collaborative quality assurance program", Clin Chem, (1986); 32, pp.845-9

d'Espagnat, B. (1983) *In search of reality* Springer-Verlag, New York

Dolk, DR. Kirsck, RA II. "A relational information resource dictionary system" Communications of the ACM (1987); 30(1) pp.48-61

Dreyfus, H.L. Dreyfus, S.E. "Making a mind versus modelling the brain: artificial intelligence back at a branchpoint." Daedalus, (Winter 1988); 117(1), pp.15-43

Feigenbaum, EA. "Knowledge processing: from file servers to knolwedge servers" Proceedings of the fourth Australian Conference on Applications of Expert Systems, (1987); pp.1-10

Feigenbaum, EA. "The art of artificial intelligence: themes and case studies in knowledge engineering". Proceedings of IJCAI-5. (1977); pp.1014-1029

Griner, PF. Mayewski, RJ. Mushlin, AI. Greenland, P. " Selection and interpretation of diagnostic tests and procedures". Ann Intern Med, (1981); 944, pp.553-92

Hayes-Roth, F. Waterman, DA. Lenat, D. "An overview of expert systems", In: *Building expert systems.* eds., Hayes-Roth, F. Waterman, DA. Lenat, D. (1983); pp. 3-29

Horn, K. Compton, P.J. Lazarus, L. Quinlan, J.R. "An expert system for the interpretation of thyroid assays in a clinical laboratory", Aust Comp J, (1985); 17, pp.7-11.

Jansen, R. Compton, P. "The knowledge dictionary, a relational tool for the maintenance of expert systems", Proc. Int. Conf. on Fifth Generation Computer systems, Tokyo (1988); pp.1159-1167

Jansen, R. Compton, P. "The knowledge dictionary: an application of software engineering techniques to the design and maintenance of expert systems" Proceedings of the AAAI'88, workshop on the integration of knowledge acquisition and performance systems, (1988)

Jansen, R. Compton, P. "The knowledge dictionary: a data dictionary approach to the maintenance of expert system" Knowledge Based Systems (1989); (2)1 pp14-26

Jansen, R. Compton, P. "The knowledge dictionary: storing different knowledge representations", (1989) *these Proceedings*

Kuhn,TS. (1969) *The structure of scientific revolutions.* University of Chicago Press, Chicago.

Lakatos, I. *Philosophical Papers* . Cambridge University Press, Cambridge

Lenat, DB. Feigenbaum, EA "On the thresholds of Knowledge" Proceedings of the fourth Australian Conference on Applications of Expert Systems, (1988); pp.31-56 (Also MCC TEchnical Report AI-126-87)

Lenat, DB. "When will machine learn?" Proc. Int. Conf. on Fifth Generation Computer systems, Tokyo (1988); pp.1239-1245

Lock Lee, LG. Teh, K. Campinini, R. "An expert operator guidance system for an iron ore sinter plant". Proceedings of the third Australian Conference on Applications of Expert Systems, (1987); pp.61-85

Lonergan BJ. (1958) *Insight* Darton, Longman and Todd, London

McCarthy,J. "Circumscription - a form of nonmonotonic reasoning" Artif Intell (1980); 13:1 pp.27-39

McLuhan, M. Parker, H. (1968) *Through the vanishing point: space in poetry and painting.* Harper and Row, New York

McNamara, AR. Lock Lee, LG. Teh, KC. "Experiences in developing an intelligent operator guidance system", Proceedings of the Australian Joint Artificial Intelligence Conference (AI'88) Adelaide (1988); pp.33-40

Newell, A. Simon, H. "Computer Science as Empirical Enquiry: Symbols and Search" reprinted in *Mind Design* ed. ed. Haugeland, J. MIT Press,Cambridge, (1981)

Piaget, J. (1929) *The child's conception of the world* Routledge and Kegan Paul, London

Popper, KR. (1963) *Conjectures and refutations* Routledge and Kegan Paul Ltd., London

Quinlan, J.R. Compton, P.J. Horn,K.A. Lazarus, L. "Inductive knowledge acquisition: a case study", In: *Applications of Expert Systems.* ed., Quinlan JR, Addison Wesley, London (1987), pp.159-73.

Rappaport AT., Gaines BR. Integrating Knowledge acquisition and performance systems Proceedings of the Australian Joint Artificial Intelligence Conference (AI'88) Adelaide, (1988); pp.317-336

Rock I. (1983) *The logic of perception* MIT Press, Cambridge, Mass

Schwartz, WB. Patil, RS. Szolovitis, P. "Artificial Intelligence in medicine: where do we stand", N. Eng. J. Med (1987); 316 pp.685-8

Shaw, MLG. "Validation in a knolwedge acquisition system with multiple experts", Proc. Int. Conf. on Fifth Generation Computer systems Tokyo (1988) pp.1259-1266

Soloway, E. Bachant, J. Jensen, K. "Assessing the maintainability of XCON-in-RIME: coping with the problems of a VERY large rule-base", Proceedings of AAAI87, Seattle, (1987); pp. 824-829

Stratton, G. "Vision without inversion of the retinal image" Psychol Rev (1897); 4 pp. 341-60, 463-81 (referred to by Kaufman L. (1979) *Perception:the world transformed*Oxford University Press, New York)

Suwa, M. Scott, AC. Shortliffe, EH. "Completeness and consistency in a rule-based system" In: *Rule-based Expert Systems,* eds., Buchanan BG and Shortliffe EH. Addison Wesley, Readingg Mass, (1984); pp.159-70.

Taylor, DAH. (letter) Nature (1984) 309(10) p108

Treitel , J. (letter) Nature (1984) 308(12) p 580

Waterman, DA. (1986) *A guide to expert systems* Addison Wesley, Reading, Mass.

Winograd, T. Flores, F. (1987) *Understanding computers and cognition: a new foundation for design.* Addison Wesley, Reading, Mass.

Wittgenstein, L. (1953) *Philosopical Investigations* Blackwell, Oxford.

KNOWLEDGE ACQUISITION: THE CONTINUUM LINKING MACHINE LEARNING AND EXPERTISE TRANSFER

Brian R Gaines

Knowledge Science Institute
Department of Computer Science, University of Calgary
Calgary, Alberta, Canada T2N 1N4
(gaines@cpsc.ucalgary.ca, (403) 220-5901)

ABSTRACT

There is currently a major paradigm split in knowledge acquisition research and practice between techniques for the transfer of existing knowledge from human experts through "expertise transfer" and those for the creation of new expertise through empirical induction or "machine learning". There is, however, a fundamental relation between the two paradigms in that existing expertise was at some time derived through empirical induction. There is also continuity between the two paradigms in that existing expertise may be partial, erroneous, and of various forms, such that it cannot completely replace empirical induction but may serve to to guide and expedite it. This paper provides a quantitative framework for knowledge acquisition that encompasses machine learning and expertise transfer as related paradigms within a single spectrum, shows that the development of practical knowledge-based systems generally lies between these extremes, and suggests system architectures that combine the underlying paradigms.

INTRODUCTION

Research and practice in advanced information systems has developed major technologies for "knowledge acquisition" without very strongly founded concepts of the nature of "knowledge" itself. In the economic literature the notion that "knowledge" in general is an abstract term for a concrete economic entity similar in status to other "goods" was extensively developed by Machlup (1980, 1982, 1984), and has found increasing acceptance in both professional (Machlup & Mansfield 1983) and popular literature (Sveiby & Lloyd 1987). It underlies the powerful metaphor that Hayes-Roth (1984) used to explain the significance of expert systems in the early 1980s, that:

> "The power is in the knowledge. Mining, molding, assembling and refining expert knowledge are the main problems."

There are strong analogies between the classical resource management and extraction industries and the new knowledge management and extraction industries which serve to guide us in developing knowledge-based industries.

There are also differences which we do not yet fully comprehend and which may mislead us if we stretch the analogies too far. One of these is the essential human role in knowledge production and application as a by-product of our social existence. The knowledge sources that we mine are already partially processed by prior human activity and will continue to be further processed in use. There are parallels in classical resource extraction and manufacturing, but the ongoing dynamics of knowledge processing dominate knowledge-based industries in such a way as to typify them. Knowledge is an essentially dynamic resource which requires active management to maintain its existence.

This paper focuses on the analogy underlying Hayes Roth's metaphor, attempting to develop a framework for knowledge acquisition that encompasses the partial processing and dynamic aspects of knowledge resources. In particular, the framework developed encompasses knowledge acquisition through both machine learning and expertise transfer.

EXPERTISE TRANSFER AND EMPIRICAL INDUCTION

There is currently a major paradigm split in knowledge acquisition research and practice between techniques for the transfer of existing knowledge from human experts and those for the creation of new expertise through empirical induction or "machine learning" (Gaines & Boose 1988, Boose & Gaines 1988). There is, however, a fundamental relation between the two paradigms in that existing expertise was at some time derived through empirical induction. There is also continuity between the two paradigms in that existing expertise may be partial, erroneous, and of various forms, such that it cannot completely replace empirical induction but may serve to to guide and expedite it (Gaines 1987a).

Figure 1 makes these relations explicit:

At the top, some "world" acts as a source from which data may be acquired.

On the left the acquisition is by a person who models the data and becomes an "expert" about some aspect of the world. In developing a knowledge-based system such experts may be interviewed using expertise transfer tools (Boose 1986) and their "knowledge" (model of part the world) may be transferred to a computer-based knowledge base.

On the right the acquisition loads a computer-based database. In developing a knowledge-based system such a database may be modeled using empirical induction tools (Quinlan 1986) and the resultant "knowledge" (model of part the world) may be transferred to a computer-based knowledge base.

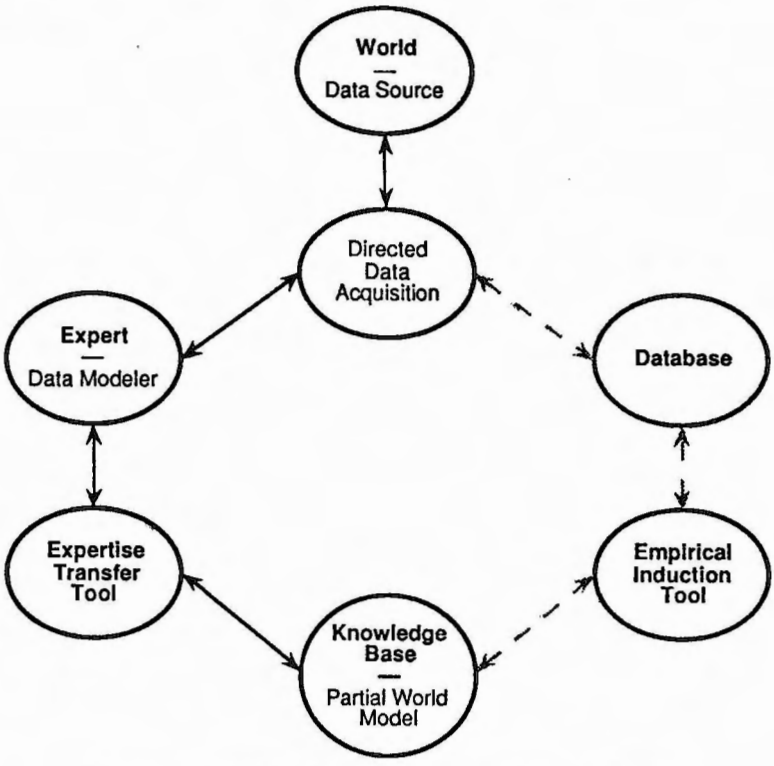


Figure 1 Expertise transfer and empirical induction

Figure 1 neatly captures the relation between expertise transfer and empirical induction. However, it over-emphasizes the separation between the two approaches:

On the left, the "expert" rarely makes available pure knowledge, that is, a minimal correct model of part of the world. Experts provide a mixture of relevant and irrelevant viewpoints, correct and incorrect modeling rules, case histories with partially known outcomes, and so on. The current generation of "expertise transfer" and "knowledge support" systems, such as Aquinas (Boose & Bradshaw 1987) and KSSO (Gaines & Shaw 1987) provide a wide range of tools to cope with, integrate, and clean up, this variety of forms and qualities of knowledge.

On the right, the "database" is not a complete or unselective collection of information about the world. It has been structured and collected by people for a purpose, and this structuring and collection already involves substantial expertise. Figure 1 emphasizes that the data used in both expertise and database formation is obtained by *directed* data acquisition. The provision of that direction is itself a task for experts.

Thus, an expert provides a mixture of knowledge, data, and garbage (incorrect or irrelevant knowledge and data), and so does a database. There is much in common between both the left- and right-hand paths in Figure 1—both expertise transfer and empirical induction involve the extraction of knowledge from partial mixes of knowledge, data, and garbage.

The following section provides a qualitative analysis of the various qualities of information between knowledge and data, and later sections provide quantitative measures of the resultant trade-off.

LEVELS OF EXPERTISE

Consider a practical knowledge-based system development in which both human experts and various sources of case histories are available, for example, Quinlan's development of thyroid diagnosis rules (Quinlan, Compton, Horn & Lazarus 1987). The experts might know:

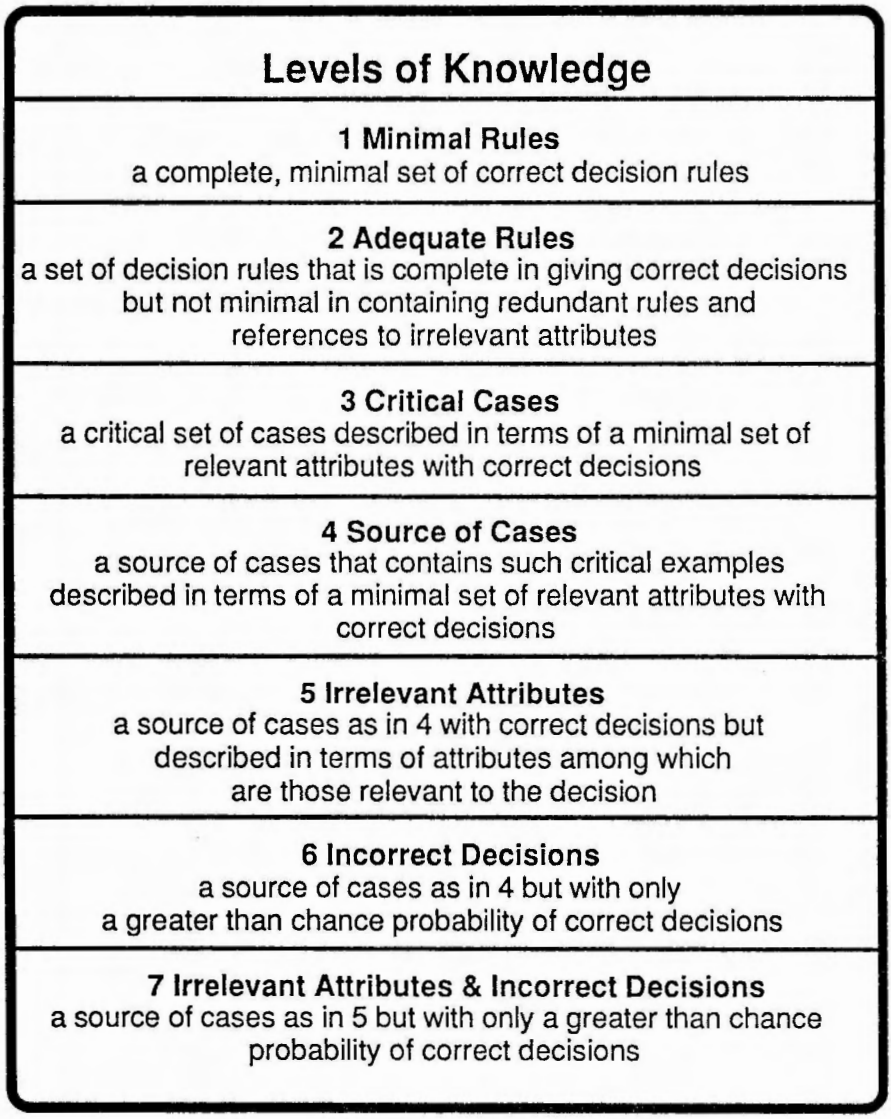


Figure 2 Levels of expert knowledge

This is a continuum of decreasing knowledge on the part of the human expert. It encompasses a range of situations met in practice, and it raises the question of how the amount of knowledge available from the expert affects the amount of data required for effective empirical induction. For case 1 no data is required for empirical induction since the correct answer is available. For case 7 the 'expert' has provided little except access to a source of data from which the correct answer might be derived. How much data is required for an optimal empirical induction procedure to derive 1 given 7, and how much less data is required for cases 2 through 6?

The following section report some studies that give quantitative answers to these questions. The studies are empirical rather than analytic so that it is not apparent yet how the answers generalize to arbitrary situations. However, they establish some base-line data which is interesting in its own right, a possible guide to practitioners of knowledge acquisition, and a test case for potential analytic estimates of the ratios involved.

THE TRADE-OF BETWEEN KNOWLEDGE AND DATA

INDUCT (Gaines 1989) is part of a knowledge acquisition tool KSS0 (Gaines 1987b) which uses entity-attribute grid techniques (Shaw 1980, Shaw & Gaines 1983, Boose 1984) to elicit relevant attributes and critical entities from experts, and empirical induction based on these to build a knowledge base in terms of classes, objects, properties, values and methods (rules). The tool also accepts rules entered by experts and entity-attribute data from databases, so that its range of knowledge/data combinations encompasses all those listed in the previous section.

INDUCT is comparable in its approach and performance to other empirical induction algorithms (Quinlan 1987), having a particularly high noise rejection—technical details are given in the Appendix. However, we believe that the experimental results on the trade-offs between knowledge and data given below are fairly independent of the empirical induction algorithm used—they are systemic consequences of the quality of knowledge available rather than side-effects of a particular approach to inductive modeling.

Cendrowska's (1987) contact lens data (Figure 3) used in her exposition of PRISM has been taken as a starting point for our trade-off studies since it is well-defined, previously analyzed and results in a range of rules of varying complexity, some of which are supported by a high proportion of the data set and others of which are supported by only single cases. The deterministic, complete data set involves 24 cases described in terms of 3 binary attributes, 1 ternary attribute and 1 ternary decision attribute. PRISM gives a solution based on 9 rules, but using default logic correct solutions with only 6 rules are available.

On this data set INDUCT generates both the 9 rule and 6 rule solutions dependent on whether default logic is allowed (exception rules are given higher priority than partially-correct default rules). Figure 4 shows the 9 rule solution—the first number after each rule is the percentage correct on the test data—it would be lower if the data were noisy—the second number is the statistical significance of the rule in terms of the percentage probability that it has arisen by chance.

Age	Prescription	Astigmatism	Tear Production	Lens
young	myope	not astigmatic	reduced	none
young	myope	not astigmatic	normal	soft
young	myope	astigmatic	reduced	none
young	myope	astigmatic	normal	hard
young	hypermetrope	not astigmatic	reduced	none
young	hypermetrope	not astigmatic	normal	soft
young	hypermetrope	astigmatic	reduced	none
young	hypermetrope	astigmatic	normal	hard
pre-presbyopic	myope	not astigmatic	reduced	none
pre-presbyopic	myope	not astigmatic	normal	soft
pre-presbyopic	myope	astigmatic	reduced	none
pre-presbyopic	myope	astigmatic	normal	hard
pre-presbyopic	hypermetrope	not astigmatic	reduced	none
pre-presbyopic	hypermetrope	not astigmatic	normal	soft
pre-presbyopic	hypermetrope	astigmatic	reduced	none
pre-presbyopic	hypermetrope	astigmatic	normal	none
presbyopic	myope	not astigmatic	reduced	none
presbyopic	myope	not astigmatic	normal	none
presbyopic	myope	astigmatic	reduced	none
presbyopic	myope	astigmatic	normal	hard
presbyopic	hypermetrope	not astigmatic	reduced	none
presbyopic	hypermetrope	not astigmatic	normal	soft
presbyopic	hypermetrope	astigmatic	reduced	none
presbyopic	hypermetrope	astigmatic	normal	none

Fig.3 Cendrowska's (1987) contact lens data

prescription=hypermetrope & astigmatism=not astigmatic & tear production=normal -> lens recommendation=soft 100% 0.904%
age=young & astigmatism=not astigmatic & tear production=normal -> lens recommendation=soft 100% 4.34%
age=pre-presbyopic & astigmatism=not astigmatic & tear production=normal -> lens recommendation=soft 100% 4.34%

tear production=reduced -> lens recommendation=none 100% 0.355%
age=presbyopic & prescription=myope & astigmatism=not astigmatic -> lens recommendation=none 100% 39.1%
age=pre-presbyopic & prescription=hypermetrope & astigmatism=astigmatic -> lens recommendation=none 100% 39.1%
age=presbyopic & prescription=hypermetrope & astigmatism=astigmatic -> lens recommendation=none 100% 39.1%

prescription=myope & astigmatism=astigmatic & tear production=normal -> lens recommendation=hard 100% 0.463%
age=young & astigmatism=astigmatic & tear production=normal -> lens recommendation=hard 100% 2.78%

Fig.4 Decision rules derived from the dataset of Figure 3

This data set is used as a kernel from which to generate corrupted data with varying probabilities of errors and with varying numbers of irrelevant attributes as shown in Figure 5. The generator selects an entity at random from the 24 cases, randomly changes the decision according to a prescribed probability, and adds a prescribed number of irrelevant binary attributes with random values. The corrupted data sets are run incrementally on INDUCT to determine the minimum amount of data necessary for a correct solution. This quantity is itself a random variable and 10 data sets generated with the same parameters are run to obtain more robust estimates of the data requirement.

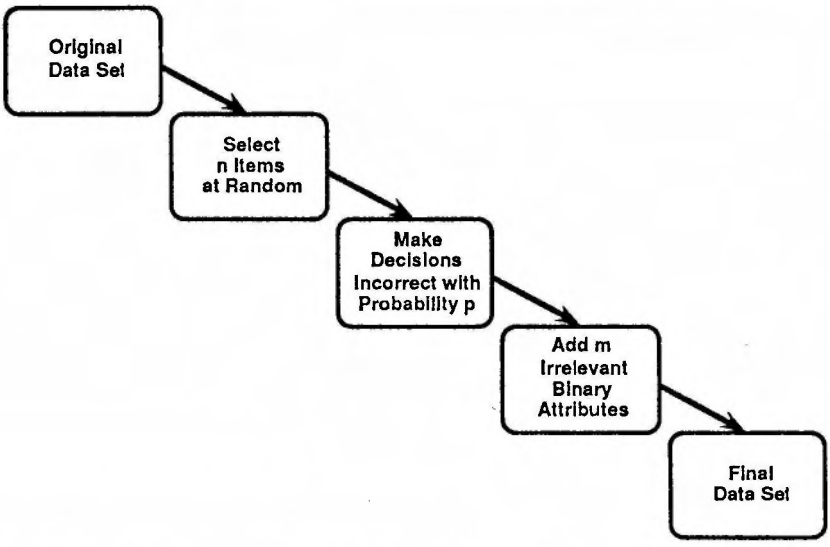


Fig.5 Dataset generation for empirical studies of knowledge/data tradeoff

Figure 6 shows the results obtained to date with a variation from 6 to over 1000 data items being needed to cover the spectrum of knowledge availability specified in cases 1 through 7 in Figure 2. It is interesting to note that very high levels of noise can be tolerated—the expert by no means has to be 100% correct, only better than chance. Noisy irrelevant attributes cause similar effects to noise in the decision—this seems to validate claims that knowledge acquisition tools targeted on eliciting relevant attributes are in themselves worthwhile. There is strong interaction between errors and irrelevancy—much more data is required to eliminate both than either alone.

Knowledge	*Data Required*	
	Mean	S.D.
Exact rules	6	0
Critical cases	18	0
Correct cases	90	43
10% Errors	123	49
25% Errors	326	159
1 Irrelevant Att.	160	77
2 Irrelevant Atts.	241	125
5 Irrelevant Atts.	641	352
10% Err. + 1 Irr.Att.	1970	1046

Fig.6 Data/knowledge tradeoff

COMBINING EXPERTISE TRANSFER AND EMPIRICAL INDUCTION

Current systems for expertise transfer generally also provide empirical induction tools for modeling cases and deriving rules from, for example, entity-attribute grid data. The studies reported in this paper give a formal framework within which the relative roles of these tools, and their interplay and integration, can be analyzed. The seven levels of expertise defined in Figure 2 are those we actually find in the "experts" typically available to us in the development of knowledge-based systems. Our "expertise transfer" methodologies give us problem-solving attributes and exemplary cases expressed in terms of those attributes. There are no guarantees that the attributes elicited are all necessary or that the cases elicited all have correct decisions. The results of the previous section show that we can eliminate irrelevant attributes and incorrect decisions providing:

(a) A sufficient set of relevant attributes for problem-solving has been elicited;

(b) an adequate number of cases has been elicited in relation to the number of irrelevant attributes and probability of error in exemplary cases.

Achieving condition (a) has been a major design requirement in even the earliest personal construct elicitation systems. Shaw's (1980) original design for a computer-based grid elicitation system emphasized features aimed at stimulating the user to add complementary attributes through feedback about those already elicited, and these have been further refined and developed in later systems.

The difficulty with achieving condition (b) is that estimates of the number of irrelevant attributes and the error rates in case data may be difficult to obtain. However, even coarse estimates make it possible to use studies such as that reported above to obtain lower bounds on data requirements.

What is interesting in considering requirement (b), however, is to note that additional case history data and improving estimates of error rates become available through the application of the knowledge-based system being developed. Whatever initial expertise is available can be used to prime an expert system that then, through use, acquires more case data on which to base improved empirical induction.

Additional cases that conform to, and hence confirm, existing rules present no problems. However, those which induce learning, that is a change in rules, also suggest that the decisions being recommended by the system are suspect. As in human behavior, the detection of anomalies between knowledge and data is critical to both action and learning.

I have previously suggested that rules based on empirical induction, such as those shown in Figure 4, should be coupled with anomaly detection rules that trigger "surprise" if their premises are true but their conclusions are false (Gaines 1988b). In such cases, further empirical induction on the enhanced data set will result in either the rule being changed or the new data rejected as in error. This is precisely the process that Stich and Nisbett (1984) have termed the expert's management of "reflective equilibrium" in their analysis of expertise using Goodman's (1973) model of induction.

The current generation of expert systems, based on taking static snapshots of expertise, will become obsolete very rapidly and present major problems of maintenance. We will be forced to move very rapidly toward what Steels (1986) has termed "second generation" systems able to learn from experience (Gaines 1988a). However, we have also to remember that the associated human experts will also be acquiring new knowledge from many sources not necessarily accessible to the knowledge-based system. Our system architectures must accomplish a long-term integration of expertise transfer and machine learning methodologies.

WHAT MAY BE ACQUIRED THROUGH COMBINED ET AND EI

The model of Figure 1 can be extended to encompass other aspects of knowledge acquisition, notably the roles of neural networks and text processing. The "directed data acquisition" at the top of the model is not readily emulated using current knowledge acquisition technologies. We may obtain labels for attributes from an experts and have them characterize cases in terms of these, but our methodologies do not capture the way in which the expert does this, the procedures for measuring the values of attributes. Thus, our knowledge-based systems cannot describe cases themselves but must question clients. In practice we assume that the clients themselves are already sufficiently expert to understand the expert's terminology and are able to supply valid characterizations of cases within the expert's conceptual framework. These are strong assumptions.

Figure 7 summarizes the levels and types of knowledge that we may regenerate given sufficient data based on lower levels of knowledge. A combination of expertise transfer and empirical induction enables us to regenerate the *prescriptive* knowledge of the expert in the form of decision-making models, the *descriptive* knowledge in the form of relevant attributes and their critical values, and the *ecological* knowledge in the form of cleaned-up case histories. What remains inaccessible is the role of the expert as an *interactive* transducer providing measurements of the world.

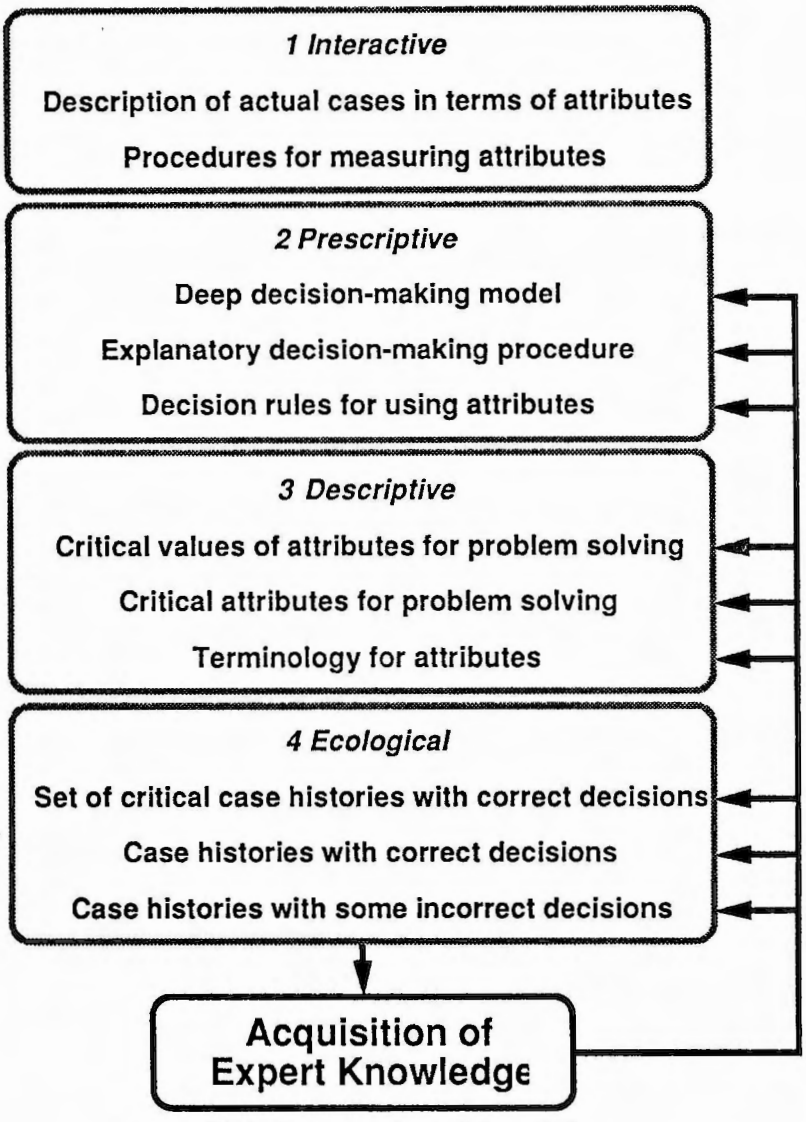


Fig.7 Regeneration of knowledge

Speculatively, we suggest that the role of neural network research within knowledge acquisition is to emulate the human capabilities of perceptual learning that enable us to form new cognitive constructs using the mass of available data to our sensations (Grossberg 1988, Eckmiller & Malsburg 1988). The lowest level of "directed data acquisition" in Figure 1 requires the formation of measurements, the recognition of new patterns, that provide values for attributes appropriate to problem solving.

This is not a new concept in terms of the direction of much research on neural networks. However, it is novel in the expertise transfer community because, so far, the emphasis has been on transferring the high level knowledge, not the perceptual capabilities. As knowledge-based systems develop it is reasonable to expect more emphasis of complete functionality, including the emulation of perceptual capabilities. It is reasonable to aim for these to be an enhancement of existing approaches rather than a complete change to radically different technologies.

Figure 8 shows Figure 1 extended with a neural network learning to emulate the directed data acquisition through perceptual learning. It also shows a second extension in that expertise is conventionally externalized and communicated through encoding in media such as text and diagrams. This provides yet another path into the knowledge base through tools that extract knowledge from such material.

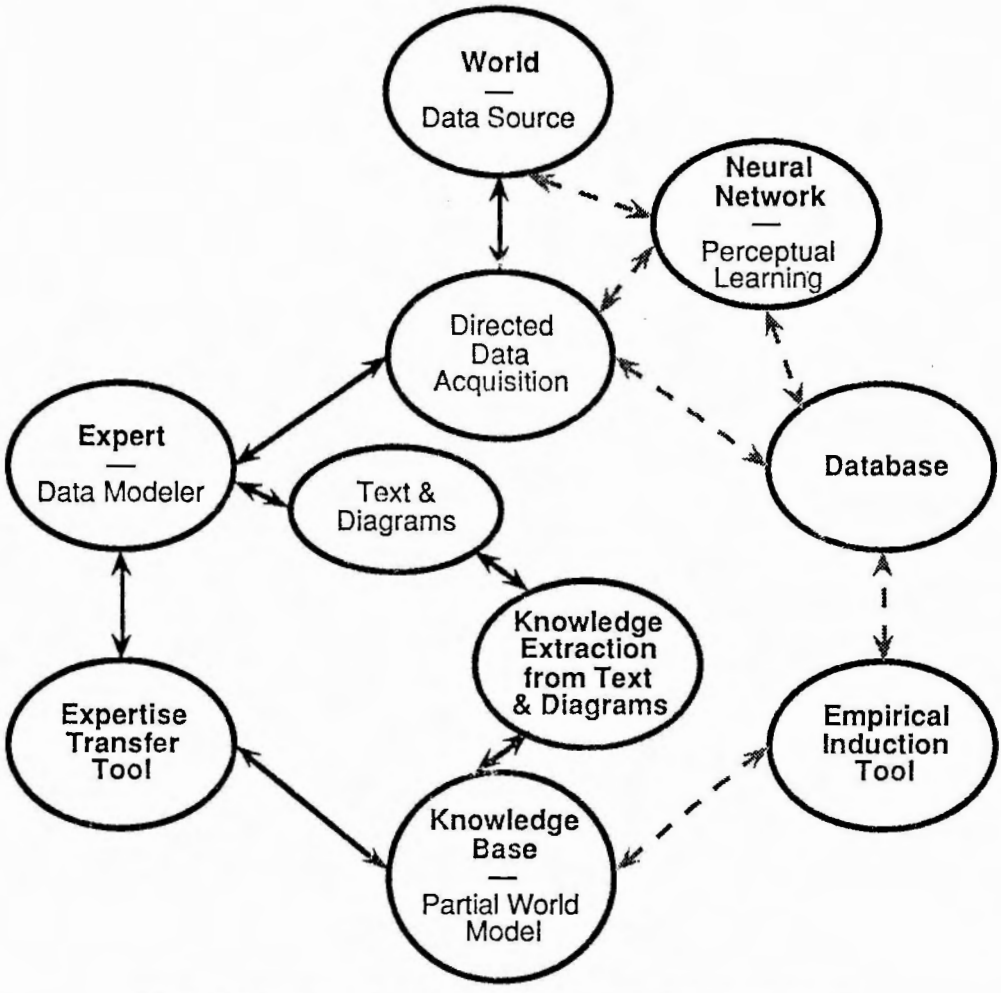


Fig.8 Integration of various forms of knowledge acquisition

A knowledge acquisition system with the architecture of Figure 8 would integrate the state of the art of four diverse artificial intelligence technologies. It is a reasonable objective for next generation research.

CONCLUSIONS

At one level the studies reported in this paper may be seen as legitimating current practice in research and development for knowledge acquisition systems. Most applicable systems are providing tools for both expertise transfer and empirical induction, and for using thse in closely coupled combinations. However, some of the tutorial literature presents them as almost competetive approaches rather than as extremes along a continuum of possibilities that match a continuum of types and levels of expertise.

In architectural terms, the analysis leading to Figure 8 provides a integrative structure for four of the major technologies supporting knowledge acquisition processes in artificial intelligence research. A practical system with such an architecture will involve many paths not shown in the figure which support the meta-tasks of managing the acquisition process, and it will involve multiple experts in different roles.

In practical terms, the studies reported in this paper provide a principled, quantitative approach to many of the different techniques being used for knowledge acquisition, from those based on expert interviews to those based on empirical induction. The trade-off data is a guide to practitioners as to the appropriate approach and data requirements in their situations—it needs testing with other data sets but that is now a matter of sheer number crunching.

In theoretical terms, the approach taken in this paper seems to offer the possibility of developing a quantitative science of knowledge in terms of the amount of data reduction that knowledge buys us when carrying out empirical induction. This can be seen as a reasonably principled economic evaluation of the knowledge.

Figure 9 shows the results of Figure 6 used to define a measure of knowledge in terms of the logarithm of the data required to achieve it. This gives a basis for an economic measure of the "value" of different forms and levels of knowledge if we assume, for example, that obtaining each item of data incurs equal cost. The studies in this paper are empirical using a particular situation. It should be possible to obtain some analytic results for more general cases.

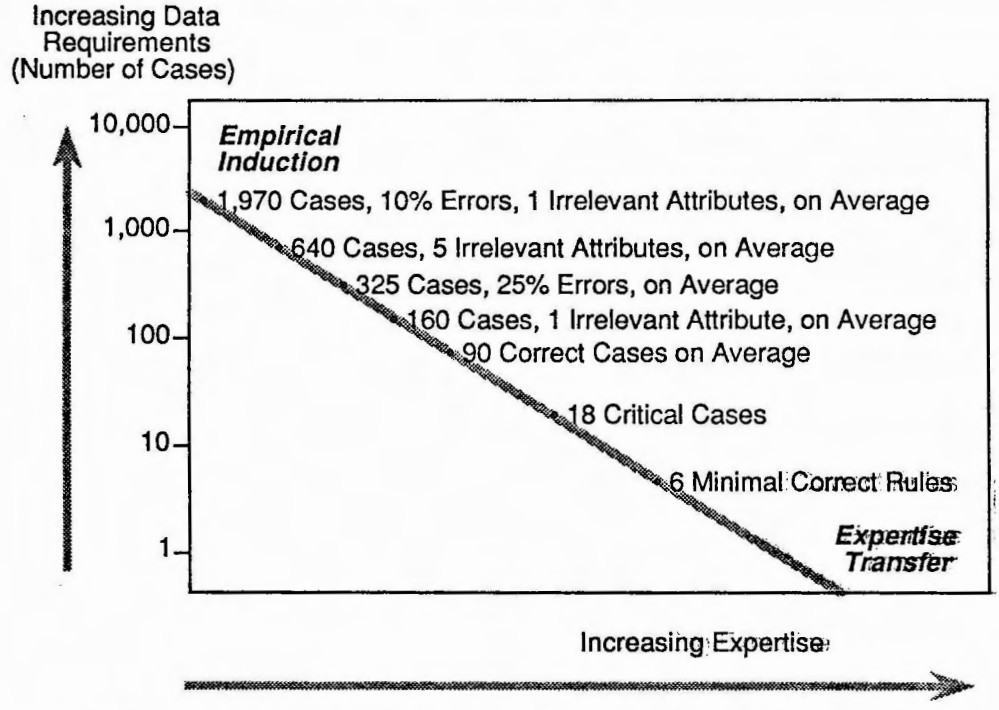


Fig.9 Knowledge/data tradeoff in expertise transfer and empirical induction

In design terms, the most important results obtained are that a single inductive algorithm is adequate to cover the complete spectrum of cases 1 through 7 of "levels of expertise" defined at the beginning of this paper, from rule simplification, to noise reduction and relevancy determination, and that it is possible to generate an integrated system that ranges across that spectrum, from the transfer of expertise from experts to the creation of equivalent expertise through empirical induction.

ACKNOWLEDGEMENTS

Financial assistance for this work has been made available by the Natural Sciences and Engineering Research Council of Canada. I am grateful to many colleagues for discussions over the years that have influenced this paper. In particular I would like to thank John Boose, Jeff Bradshaw, Mildred Shaw, and Ross Quinlan, for access to their own research and many stimulating discussions.

APPENDIX: THE INDUCT ALGORITHM

It is relevant to the assessment of the results in this paper to understand the method used to derive decision rules from data, and this section outlines the INDUCT algorithm, linking it to previous approaches, clarifying enhancements made and their roles in the studies, and comparing performance with the best known methods.

Cendrowska (1987) has shown that empirical induction through decision trees and direct conversion to rules, even with pruning, leads to rule sets that test the values of irrelevant attributes and are much larger than is necessary. Her PRISM algorithm goes from entity-attribute data direct to rules but does not address the problems of noisy data or missing values. Quinlan (1987) has developed extended pruning techniques in a way that are effective in coping with noisy data but still involve decision tree production with problems of irrelevant attributes and missing values.

INDUCT extends the PRISM algorithm to control direct rule generation through statistical tests that are effective in dealing with both noisy data and missing values. Figure 10 shows the basis for these statistical tests. Given a universe of entities, E, a target predicate, Q, and a set of possible test predicates of the form, S, on entities in E, use them to construct a set of rules from which the target predicate may be inferred given the values of the test predicates.

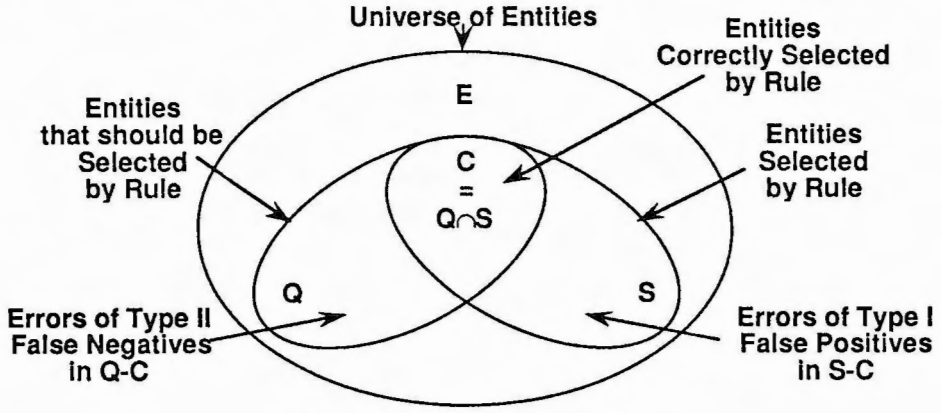


Fig.10 Problem of empirical induction

For the purposes of the statistical analysis the forms of S and Q do not matter. One may regard S as a *selector* choosing those e out of some subset of E for which to assert Q(e), and compare the selection process of the rule with that of random selection, asking "what is the probability that random selection of the same degree of generality would achieve the same accuracy or greater."

This probability is easily calculated, let Q be the relevant entities in E for which Q(e) holds, S be the selected entities in E for which S(e) holds, C be the correct entities in E for which both S(e) and Q(e) hold:

$$Q \equiv \{e: e \in E \wedge Q(e)\} \quad (1)$$

$$S \equiv \{e: e \in E \wedge S(e)\} \quad (2)$$

$$C \equiv \{e: e \in E \wedge S(e) \wedge Q(e)\} \quad (3)$$

Let the cardinalities of E, Q, S and C be e, q, s and c respectively. The probability of selecting an entity from E for which Q holds at random is:

$$p = q/e \quad (4)$$

The probability of selecting s and getting c or more correct at random is:

$$r = \sum_{i=c}^{s} {}^{s}C_i \, p^i \, (1-p)^{s-i} \quad (5)$$

The advantage of using r as a measure of the correctness of a rule is that is easily understood, as the probability that the rule could be this good at random, and that it involves no assumptions about the problem such as sampling distributions. Note that if c=s (all correct) then log(p) = s log(q/u) which seems to be the basis of 'information-theoretic' measures.

The probability that the performance of an isolated rule could be obtained by random selection is not adequate in itself to evaluate a rule that has itself been selected as best after a search through many possible rules. If n different rules have been searched then we may ask "what is the probability that the rule found by search achieved its performance by chance," that is the probability that a rule will have been found that in itself has a probability of being selected at random of r. This probability is:

$$t = 1 - (1-r)^n \qquad (6)$$

As one would expect this approximates to nr if this term is substantially less than 1.
To obtain a value for t one needs to estimate n, the number of rules in the search space, and this requires information about the forms of possible rules. If the rules are conjunctions of value tests of attributes as illustrated in (3) then n_1 can be estimated for rules with one clause, n_2 for rules with two or less clauses, and so on. For example if there are m attributes the i'th of which can be tested in m_i possible ways, then:

$$n_1 = \sum_{i=1}^{m} m_i \qquad (7)$$

$$n_2 = (n_1)^2 - \sum_{i=1}^{m} (m_i)^2 \qquad (8)$$

Missing values are taken into account by assuming that they might have any value. When the selection of an entity is tested a missing value is assumed to have the required value for selection. In the statistics a selection based on missing values is allowed to contribute to false positives but not to correct positives. This has important consequence in knowledge acquisition since it allows the expert to enter conjunctive rules as if they were entities with missing values. INDUCT then generates the same or an equivalent smaller rule set. It is reasonable to test the consistency of an inductive procedure by requiring the rule-set produced by it to be 'fixed-point' if re-entered as data.

INDUCT has been tested on a wide range of data sets in the literature together with many artificial data sets with known degrees of noise, missing values and irrelevant attributes, and found to perform consistently at least as well as the previously published best results. For example, with Quinlan's (1987) very noisy data sets, Prob-Disj (disjunction of three terms in three attributes, one irrelevant attribute, 10% noise), and Digits (7-segment display, 10% noise in attributes, Bayes optimal solution 26% errors) the comparisons are shown in Figure 11.

Results on Prob-Disj Data (3-term disjunct)

Data	Error Rate (Type I + Type II)	
	ID3 to Pruned Rules	INDUCT
Sample	4.2 rules	3 rules
Test1	10.0%	10.0%
Test2	10.0%	10.0%

Results on Digits Data

Data	Rules	Error Rate (Type I + Type II)	
		ID3 to Pruned Rules	INDUCT
Sample	10		31.05%
	16		26.8%
Test1	10		32.0%
	16	31.3%	29.0%
Test2	10		31.4%
	16	28.3%	27.8%

Fig.11 Comparison of INDUCT and optimally-pruned ID3

REFERENCES

Boose, J.H. (1984). Personal construct theory and the transfer of human expertise. **Proceedings AAAI-84**, 27-33. California: American Association for Artificial Intelligence.

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

Boose, J.H. & Gaines, B.R., Eds. (1988). **Knowledge Acquisition Tools for Expert Systems**. London, Academic Press.

Boose, J.H. & Bradshaw, J.M. (1987) Expertise transfer and complex problems: using AQUINAS as a knowledge acquisition workbench for knowledge-based systems. **International Journal of Man-Machine Studies 26**, 3-28.

Cendrowska, J. (1987) An algorithm for inducing modular rules. **International Journal of Man-Machine Studies 27** (4), 349-370 (October).

Eckmiller, R. & Malsburg, C., Eds. (1988). **Neural Computers**. Berlin: Springer.

Gaines, B.R. (1987a) An overview of knowledge acquisition and transfer. **International Journal of Man-Machine Studies 26** (4), 453-472 (April).
Gaines, B.R. (1987b). Rapid prototyping for expert systems. Oliff, M., Ed. **Proceedings of International Conference on Expert Systems and the Leading Edge in Production Planning and Control**. pp.213-241. University of South Carolina.
Gaines, B.R. (1988a). Second generation knowledge acquisition systems. **Proceedings of the Second European Workshop on Knowledge Acquisition for Knowledge-Based Systems (EKAW'88)**. GMD-Studien Nr.143. pp.17-1-17-14. Bonn, Germany: Gellsellschaft für Mathematik und Datenverarbeitung (June).
Gaines, B.R. (1988b). Positive feedback processes underlying the formation of expertise. **IEEE Transactions on Systems, Man & Cybernetics**, SMC-18(6), 1016-1020 (November).
Gaines, B.R. (1989) An Ounce of Knowledge is Worth a Ton of Data: Quantitative Studies of the Trade-Off between Expertise and Data based on Statistically Well-Founded Empirical Induction. **Proceedings of 6th International Workshop on Machine Learning**, Cornell University, Ithaca.
Gaines, B.R. & Boose, J.H., Eds. (1988). **Knowledge Acquisition for Knowledge-Based Systems**. London, Academic Press.
Gaines, B.R. & Shaw, M.L.G. (1987). Knowledge support systems. **ACM MCC-University Research Symposium**. Austin, Texas: MCC. pp.47-66.
Goodman, N. (1973). **Fact, Fiction and Forecast**. Indianapolis: Bobbs-Merrill.
Grossberg, S., Ed. (1988). **Neural Networks and Natural Intelligence**. Cambridge, Massachusetts: MIT Press.
Hayes-Roth, F. (1984). The industrialization of knowledge engineering. Reitman, W., Ed. **Artificial Intelligence Applications for Business**. pp. 159-177. Norwood, New Jersey: Ablex.
Machlup, F. (1980). **Knowledge and Knowledge Production**. Princeton University Press.
Machlup, F. (1982). **The Branches of Learning**. Princeton University Press.
Machlup, F. (1984). **The Economics of Information and Human Capital**. Princeton University Press.
Machlup, F. & Mansfield, U. (Eds.) (1983). **The Study of Information**. New York: Wiley.
Quinlan, J.R. (1986) Induction of decision trees. **Machine Learning 1**(1) 81-106 (March).
Quinlan, J.R. (1987) Simplifying decision trees. **International Journal of Man-Machine Studies 27** (3), 221-234 (September).
Quinlan, J.R., Compton, P.J., Horn, K.A. & Lazarus, L. (1987) Inductive knowledge acquisition: a case study. Quinlan, J.R. (Ed.), **Artificial Intelligence and Expert Systems**. pp.157-173. Sydney: Addison-Wesley.
Shaw, M.L.G. (1980). **On Becoming a Personal Scientist**. London: Academic Press.
Shaw, M.L.G. & Gaines, B.R. (1983). A computer aid to knowledge engineering. **Proceedings of British Computer Society Conference on Expert Systems**, 263-271 (December). Cambridge.
Steels, L. (1986). Second generation expert systems. Bramer, M.A. (Ed). **Research and Development in Expert Systems III**. pp.175-183. Cambridge: University Press.
Stich, S.P. & Nisbett, R.E. (1984). Expertise, justification and the psychology of inductive reasoning. Haskell, T.L., Ed. **The Authority of Experts**. pp. 226-241. Bloomington, Indiana: Indiana University Press.
Sveiby, K.E. & Lloyd, T. (1987). **Managing Knowhow**. London: Bloomsbury.

A Framework for Knowledge Acquisition

Georg Klinker
Digital Equipment Corporation
DLB5-3/E2
290 Donald Lynch Blvd.
Marlboro, Massachusetts, 01752

Abstract

A wealth of knowledge acquisition tools have been developed over the past several years. They address different issues in the development of knowledge-based systems. This paper organizes the development of expert systems into several different stages representing a life-cycle for knowledge-based systems. We argue, that each phase in the development of expert systems requires different skills from a knowledge engineer. Therefore, the life-cycle for knowledge-based systems makes explicit the different kinds of support required in building expert systems. This paper further organizes existing knowledge acquisition tools around the support they provide to the knowledge engineer with different phases in the life-cycle. We conclude with a proposal -- based on the life-cycle for knowledge-based systems -- that represents a first step towards an integrated framework for knowledge acquisition.

1. Introduction

Human experts perform tasks that are typically within a narrow problem domain, but require years of special training and education to master. Over the past years AI techniques have been developed to assist human experts with their tasks. Among those AI techniques are knowledge-based systems or expert systems: computer programs that use expert knowledge. Existing expert systems have proven that AI techniques can be used to solve a variety of knowledge intensive problems. For example PROSPECTOR [Duda 84] assists geologists in mineral exploration, R1 [McDermott 82] (now called XCON) configures computer systems, and DENDRAL [Buchanan 78] determines the topological structure of organic compounds. The assistance these and other systems provide ranges from solving small pieces of a human expert's task to independently performing the complete task.

But the development of a knowledge-based system is a time-consuming and complex process. The central issue is how to develop and maintain a large and growing knowledge base, such, that the demands placed on knowledge engineers and domain experts are reduced.

A commonly proposed solution is the development of knowledge acquisition tools. A knowledge acquisition tool typically interacts with human experts and/or knowledge engineers. It organizes the knowledge it acquires and assists with generating, testing, and updating an expert system. Figure 1-1 presents an overview of the interdependencies between knowledge acquisition tools and the generated

expert systems[1]. A knowledge engineer or domain expert uses a knowledge acquisition tool to generate a knowledge base. A problem-solving method operates on the knowledge base and produces a solution. The knowledge base plus the problem-solving method form an expert system. The performance of the expert system provides feedback to the knowledge engineer or domain expert. Whenever an expert system reveals inadequacies, the knowledge engineer or domain expert uses the knowledge acquisition tool to improve it. If individual task requirements cannot be realized with the knowledge acquisition tool, the knowledge engineer manually enhances the expert system.

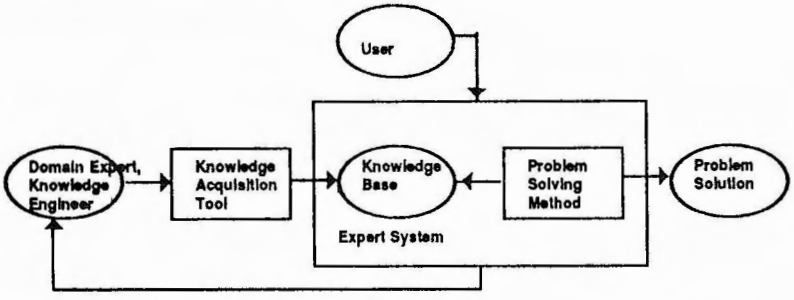


Figure 1-1: Overview

A variety of knowledge acquisition tools have been developed over the past years. They address different issues in the development of knowledge-based systems making it very difficult to compare and evaluate approaches and results. This paper organizes the development of expert systems into several different stages representing a life-cycle for knowledge-based systems. We argue, that each phase in the development of expert systems requires different skills from a knowledge engineer. Each phase represents a different problem for knowledge acquisition. Therefore, the life-cycle for knowledge-based systems makes explicit the different kinds of support required in building expert systems. Section 2 introduces a life-cycle for knowledge-based systems. Sections 3 through 7 provide more detail on each phase of the life-cycle for knowledge-based systems, point out the support required from a knowledge acquisition tool, and categorize some existing knowledge acquisition tools based on the support they provide to the knowledge engineer in different phases of the life-cycle. Section 8 concludes with a proposal -- based on the life-cycle for knowledge-based systems -- that represents a first step towards an integrated framework for knowledge acquisition.

2. A Life-Cycle for Knowledge-Based Systems

During the past decade, a wealth of expert systems have been developed. Experience in building these systems shows that development can be organized into several distinct phases, resulting in a life-cycle for knowledge based systems. Figure 2-1 illustrates the life-cycle. This description of the knowledge engineering process is based on models presented in [Motta 88] and [Hayes-Roth 83].

Rectangles represent activities a knowledge engineer must perform to build a knowledge-based system. The arrows along the center of the rectangles indicate the logical order in which those activities are performed. The life-cycle phases are highly interdependent; the results of one phase might require the

[1]This model is adapted from Dietterich's simple model of learning systems [Dietterich 82].

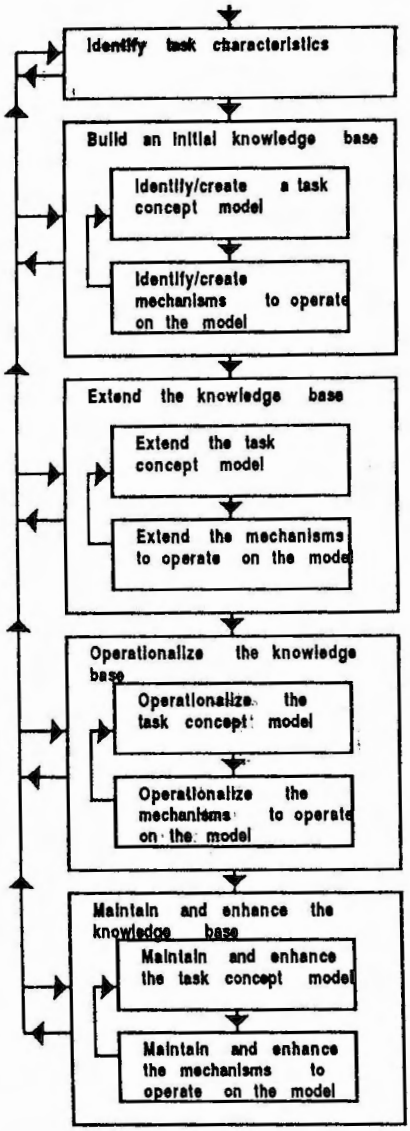


Figure 2-1: A Life-Cycle for Knowledge-Based Systems

revision of results of other phases. The iterative nature is represented by the arrows on the left of the rectangles in Figure 2-1. Each phase indicates a conceptually different task. The inner boxes refine the life-cycle phases and are highly interdependent. For example, a change in the task concept model is likely to require an update of the mechanisms that operate on the model. The outer boxes are still interdependent, but less so than the inner boxes.

As illustrated by the life-cycle model, a knowledge-based system progresses through a number of development stages. Each phase requires different skills from the knowledge engineer and, therefore, represents a different problem for knowledge acquisition. Most stages in the development process are quite complex and not well understood. Therefore, researchers have focussed on specific parts of the knowledge engineering task. No single tool exists to support the development of a knowledge-based system throughout its life-cycle. In the following sections we describe the life-cycle phases in more detail, point out the support required from a knowledge acquisition tool, and present a representative (not exhaustive) list of existing knowledge acquisition tools categorized with regard to the life-cycle model. See [Boose 88] for a more comprehensive compilation of existing knowledge acquisition tools.

The following sections are organized around an actual example: The application task to be automated is to present the design of an electromechanical system, such, that it can be evaluated for its nuclear hardness. Nuclear hardening requires the use of specific engineering-design practices to increase the resistance of an electromechanical system to the environmental effects generated by the detonation of a nuclear weapon. Designers of electromechanical systems usually have little or no knowledge about the specialized analytical methods and engineering practices of the hardening domain. The task is to assist a designer in presenting a given design of an electromechanical system in such a way that hardening experts have readily available the information they need to evaluate the system from a hardening perspective. In the following we will refer to this application task as "Write a Design Parameters Report". The system automating the task is called DPR WRINGER.

3. Identifying Task Characteristics

The first step in building expert systems is to identify the task characteristics. For each task a multitude of different characteristics can be defined. The problem is to find the ones that matter. For a user of an expert system it is critical that the system provides the expected output and only requires input readily available to the user. Therefore, relevant task characteristics include a description of a task's output and input.

For example, relevant characteristics for the task "Write a Design Parameters Report" are:

- **The resulting document is structured**, i.e., the outline of the report, the report structure, is common to a large set of applications. A report is defined by its structure (i.e., chapters, sections, subsections, etc.), fixed text components (i.e., text that always will be part of the document), and variable text components (i.e., text specific to an application). For a structured document the report's structure is common to a large set of applications. Only variable text components are specific to an individual application. For some tasks a small set of highly structured reports exists or it is possible to systematize a task into a structured document. But this is not always the case. Some tasks require a large set of reports whose structure differs from application to application. The variable text components, fixed text components, and even the structure of a document might change with every application, resulting in unstructured reports.
- **The information presented in the document is structured**, i.e., only instances of the presented information change from application to application. If only a few different concepts are used in a document, the task specific information is structured. However, different applications might still use many different instances of the same concept. If many different concepts are used, the task specific information is unstructured.

Although some progress has been made on how to characterize application tasks, the area is still poorly understood. As of today, no taxonomy exists that characterizes real life tasks. Therefore, no tool exists to assist a knowledge engineer or domain expert with identifying task characteristics. This comes as no

surprise as we do not yet have enough data points. Compared to the number of real life tasks, few expert systems have been developed so far. Only very few of those systems have been analyzed with respect to the chracteristics of the tasks they can solve. However, some work has been done to identify task criteria for some classes of tasks, e.g., reporting tasks [Klinker 88a], skeletal plan refinement tasks [Friedland 85], and classification tasks [Clancey 84].

4. Building an Initial Knowledge Base

The next step in the development process is to build an initial computational model of the domain. This includes creating a task concept model and mechanisms to operate on the model.

4.1. Creating a Task Concept Model

A task concept model or domain model consists of a set of structured terms, concepts, and interrelationships between concepts that human experts use to perform their task. Typically, building the model is done in two steps: building a structural model of the domain and extending it into a functional model.

4.1.1. Creating a Structural Model

In the preliminary phase, the knowledge engineer creates a taxonomy of the terms and concepts experts use to perform their task. This results in a structural model of the domain representing the expert's vocabulary.

Figure 4-1 demonstrates a part of the structural model for the task "Write a Design Parameters Report".

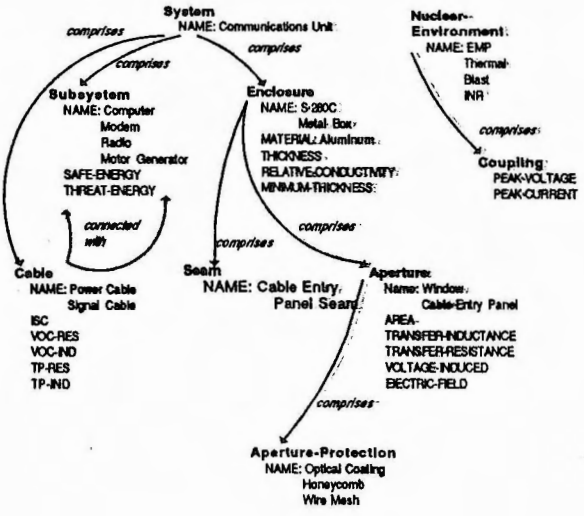


Figure 4-1: Structural Model

Tools exist that interrogate the domain expert to acquire unstructured data representing relevant terms, concepts, and their interrelations. These raw data are then structured using some methodology. Examples of such tools are ETS [Boose 84] and KSSO [Shaw 87]. Both tools use the repertory grid technique to impose some structure on the data.

4.1.2. Creating a Functional Model

To "make sense" of the terms and concepts, their interrelationships need to be defined. This includes a description of how parameters are obtained and propagated, which concepts explain other concepts, what concepts are justifications for other concepts, etc. The result is a functional model of the domain.

Figure 4-2 demonstrates a portion of the functional model for the task "Write a Design Parameters Report". The nodes describe procedures for obtaining parameters. The links, indicated by dotted lines, define the way values are propagated through the network.

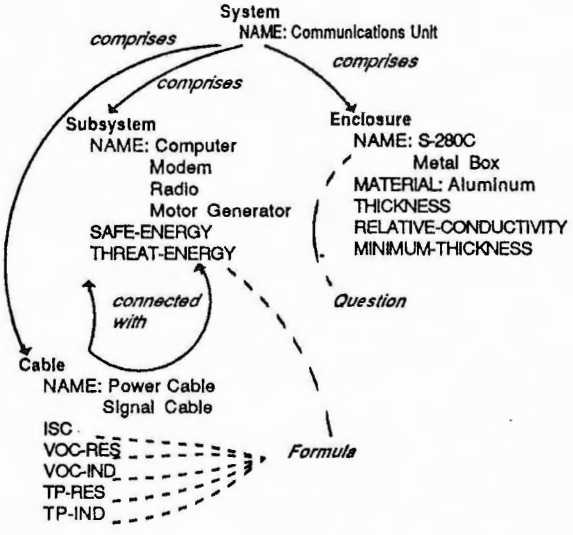


Figure 4-2: Functional Model

Tools exist that interpret data readily available, such as written text. For example, KRITON [Diederich 87] and SOK [Linster 88] perform a statistical text analysis to extract a functional domain model from pieces of text. KEATS-2 [Motta 88] provides a graphical interface for the same purpose.

4.2. Creating Task Mechanisms

Once the task characteristics are defined they can be used to identify the problem-solving mechanisms operating on the task concept model to perform a part of the task. The mechanisms then have to be configured into a problem-solving method performing the application task. The result of this phase is a computational model of the domain. That model might be common to a class of application tasks or it might be specific to the particular problem at hand. It is a model of the problem-solving behavior of the

expert system the knowledge engineer is expected to build. Building a computational model of the domain, is one of the most challenging tasks for the knowledge engineer.

Examples of mechanisms that can be used to solve parts of the "Write a Design Parameters Report" task are:

- Identify relevant information
  1. Choose an appropriate part of the report to be produced.
  2. Identify all pieces of information required for this part of the report.
  3. Determine the pieces of information which are appropriate to be elicited now.
  4. Select one piece of information to be elicited.
- Acquire information
  1. Determine a strategy to gather the selected pice of information.
  2. Apply that strategy.
  3. Integrate the pieces of information with the existing information.
- Present information in the form of a document
  1. Detemine the report structure.
  2. Select the report fragments that can be instantiated with the information into that report skeleton and instantiate them.
  3. Print the report.

These mechanisms are configured into the "acquire-and-present" problem-solving method performing the application task:

1. Identify relevant information; if done, continue with step 3.
2. Acquire information; continue with step 1.
3. Present information in the form of a document.

Some researchers concentrate on developing a general taxonomy of problem-solving mechanisms [Chandrasekaran 87, McDermott 88]. Others develop tools to help creating task mechanisms. Those make different assumptions about the degree to which a knowledge engineer has to be involved.

- **Workbenches for the Knowledge Engineer:** Some systems integrate a variety of methodologies and tools for the development of expert systems into a workbench for a knowledge engineer, providing an effective programming environment. An example is SEAR [van de Brug 86]. A different approach is programming methodologies for knowledge-based systems, such as RIME [Bachant 88].
- **Automated Tools:** Some knowledge acquisition tools try to automate the knowledge acquisition process. An automated knowledge acquisition tool typically interacts with domain experts directly. No knowledge engineer is necessary to translate the expert's knowledge into production rules. The critical feature of an automated knowledge acquisition tool is that a domain expert can use it without having to know about programming in general and specific AI techniques. An example is SOAR [Laird 87]. It advocates a general problem-solving method that specializes itself for a particular task.

5. Extending the Knowledge Base

The knowledge base must be filled with content knowledge. The initial knowledge gathered from the expert must be extended to reflect most of the knowledge used by the domain expert when solving a task. Again, extending the knowledge base includes extending the task concept model and the mechanisms to operate on the model.

5.1. Extending the Task Concept Model

Extending the task concept model includes adding new facts to the initial model from the previous phase. This is a process of "scaling-up" the knowledge base to the large size necessary for a high performance system. For example, XCON has a knowledge-base of over 10000 rules and is growing daily [Barker 89].

The follwing is an example of how the knowledge base for the DPR WRINGER can be extended with a piece of knowledge defining a question strategy for the ENCLOSUREs of a SYSTEM. The knowledge is acquired using the KNACK [Klinker 88b] tool -- the tool developed to extend the task concept model for a class of reporting tasks[2,3]

```
Which strategy can be used to determine ENCLOSUREs of a SYSTEM?

[question, inference, table, menu, graphics, formula, database,
postpone, quit] [ QUESTION ]: <cr>

question text....: Please list the enclosures

possible answers.. [ INCOMPLETE-SET, S-280C, METAL BOX ] <cr>
default answer.... [ S-280C ]: unknown
status............ [ NOT-MANDATORY ]: <cr>
```

Most of the work in knowledge acquisition tools focusses on extending the task concept model. A wide variety of tools have been developed. A major distinguishing feature between tools is the degree to which a knowledge engineer is to be involved with the acquisition process. Two classes of tools exist: workbenches and automated tools.

- **Workbenches for the Knowledge Engineer:** Some systems integrate a variety of methodologies and tools for the development of expert systems into a workbench for a knowledge engineer. Some examples of existing knowledge acquisition tools are AQUINAS [Boose 87] and TDE [Kahn 87].
- **Automated Tools:** Some knowledge acquisition tools try to automate the knowledge acquisition process. General tools and specialized tools are two approaches to automating knowledge acquisition.
    - **Specialized Tools:** Specialized tools derive their power by presupposing the problem-solving method of the expert systems they generate [McDermott 88, Gruber 87]. A problem-solving method establishes and controls the sequences of actions required to perform some task. This control knowledge dynamically defines the order in which subtasks have to be solved to perform the overall task. It also defines the kind of domain specific knowledge that is applicable within each step, thereby making explicit

---

[2]KNACK is suitable for a class of reporting tasks that include our example task "Write a Design Parameters Report".

[3]KNACK's prompts appear in normal typeface, the user's responses are underlined. A word in brackets at the end of KNACK's prompts is the default response. The user may reply with the default by hitting return: <cr>.

the different roles knowledge plays. The granularity of the problem solving method is determined by the requirement that the knowledge represented by a knowledge role can be applied without further control knowledge, i.e., the time at which that knowledge will be brought to bear is determined by its role. Thus, the problem solving method helps to identify and classify the domain knowledge by making the roles knowledge plays in the problem solution approach explicit. Roles of knowledge are a powerful means to organize the knowledge base.

Examples of automated task-specific knowledge acquisition tools are: ASK [Gruber 88], MORE [Kahn 85], and MOLE [Eshelman 87] for selective tasks; BURN [Marques 88] for sizing tasks; SALT [Marcus 88] and CGEN [Birmingham 88] for constructive tasks; KNACK [Klinker 88b] for reporting tasks; and OPAL [Musen 88] for skeletal-plan refinement tasks.

- **General Tools:** General tools pre-suppose no problem-solving mechanism, and, therefore can be used for a variety of expert systems. The impetus for research in this area is the development of multi-functional knowledge-bases, that are intended for use by a variety of different problem-solving systems. Example tools include CYC [Lenat 86] and KI [Murray 88].

5.2. Extending the Task Mechanisms

If the growing knowledge base reveals inadequacies with the mechanisms operating on the model, the mechanisms have to be updated. This includes specializing or refining a mechanism, e.g., adapting a general mechanism for skeletal-plan refinement tasks to administer cancer therapy; changing a mechanism, e.g., changing the problem-solving behaviour of a mechanism by adding or deleting a step; or replacing a mechanism that cannot accommodate the specific problem-solving requirements of the growing knowledge base, e.g., a depth-first search mechanism might have to be replaced with a faster mechanism if the search space grows larger.

For example, the problem-solving behaviour of the DPR WRINGER's Acquire information mechanism had to be changed by adding a step to check the acquired pieces of information for synonyms and errors. The updated mechanism is shown below:

- Acquire information
  1. Determine a strategy to gather the selected pice of information.
  2. Apply that strategy.
  3. Check the pieces of information for synonyms and errors.
  4. Integrate the pieces of information with the existing information.

Some tools exist to specialize a general mechanism applicable in a domain for a particular task in that domain. Examples are PROTEGE [Musen 88] for skeletal-plan refinement tasks and STUDENT [Gale 87] for statistical tasks. These tools customize general task mechanisms used in the application domain for the particular task at hand.

6. Operationalizing the Knowledge Base

The previous steps result in a task concept model and mechanisms operating on the model represented in the form of some intermediate representation. This format must be made operational, that is, transformed into a computer executable form. This includes implementing the task concept model and the mechanisms operating on the model. The implementation results in a prototype expert system.

For example, the DPR WRINGER is implemented in OPS5. The following is an english translation of the rules operationalizing the part of the task concept model that realizes the piece of knowledge used as an example in section 5.1. Rule 1 verifies that a question strategy can be used to gather a specific piece of information and asks the question. Rule 2 checks whether the result of applying a strategy is valid.

```
Rule 1:

If   the goal is to identify strategies, and
     the subgoal is to determine the NAME of an ENCLOSURE, and
     a SYSTEM with some NAME is known,
then create a request to determine the NAME of an ENCLOSURE
         using a QUESTION strategy, and
     create the question "Please list the enclosures", and
     classify the answer as NOT-MANDATORY.

Rule 2:

If   the goal is to validate a strategy result, and
     the NAME of an ENCLOSURE was determined, and
     it is not S-280C or METAL BOX,
then mark the result as POSSIBLY INCORRECT.
```

Specialized programming languages and expert systems shells have been developed to implement knowledge-based systems. OPS5 [Forgy 81] and LISP [Steele 84] are two examples of widely used programming languages. LOOPS [Bobrow 83] and KEE [Kunz 84] are only two examples of a multitude of commercially available shells.

7. Maintaining and Enhancing the Knowledge Base

Maintaining and enhancing the expert system (task mechanisms as well as task concept model) is a process that continues throughout the systems life time, and is a part of every phase in the life-cycle. Providing maintenance/enhancement support should therefore be an integral part of every knowledge acquisition tool mentioned so far. But this aspect is often neglected in existing tools.

8. A Proposal

The previous sections organized the development of expert systems into several different stages resulting in a life-cycle for knowledge-based systems. We showed that each phase requires different skills from a knowledge engineer. Therefore, the life-cycle for knowledge-based systems explicates the different kinds of support required in building expert systems. We now conclude with a proposal -- based on the life-cycle for knowledge-based systems -- that represents a first step towards an integrated framework for knowledge acquisition. The proposal outlines the research strategy of the Artificial Intelligence Research Group at Digital Equipment Corporation. It is by no means complete and is meant to serve as the basis for further discussion.

Our goal is to make the task of automating applications easier, i.e., to require less time and less competence from a developer. A developer is a person who wants to automate a given task. He understands the task and can perform it, but does not necessarily know how to program. Our proposal is based on the assumption that a developer can describe the required solution for a task (the output) and the information available to the problem-solver (the input), but not the computational mechanisms used to perform the task. Our hypothesis is that computational mechanisms can be defined on a level of

abstraction, such, that they can be indexed by the required task output and the input available. Given that assumption and hypothesis we propose an integrated framework for knowledge acquisition based on the life-cycle for knowledge-based systems. Figure 8-1 summarizes our proposal.

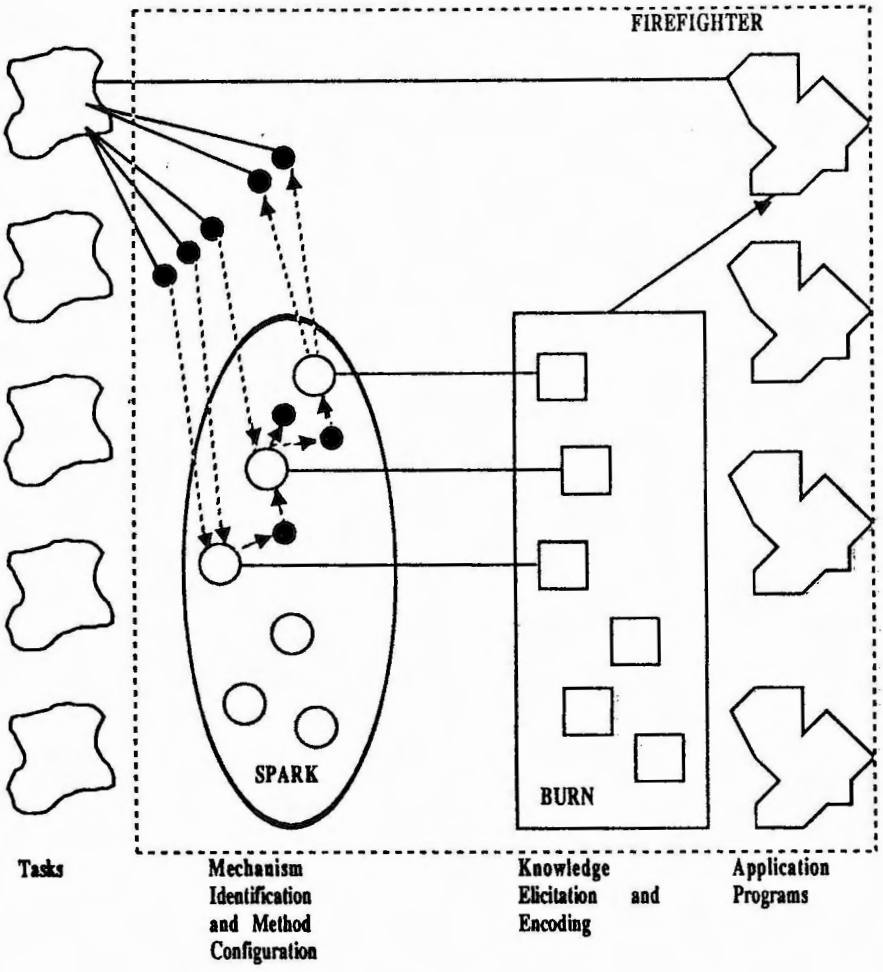


Figure 8-1: An Integrated Framework for Knowledge Acquisition

A developer wants to automate a given task. He interacts with a system, called SPARK, to **identify task characteristics**, i.e., determine the required output and input available.

SPARK then **identifies task mechanisms**. Each task mechanism solves a part of the task. The complete task can be solved by combining the mechanisms into a problem-solving method. Thus, identifying the task mechanisms involes determining the mechanisms and configuring them into a problem-solving method. Task mechanisms are identified by mapping output characteristics onto the computational mechanisms SPARK knows about. This involves identifying the mechanisms that produce the required

output. If the identified mechanisms consume input not available to the problem-solver, SPARK identifies mechanisms that produce that input. This process continues until a collection of mechanisms is found that will only use the available input to produce the required output. Once SPARK identifies the mechanisms, it configures them into a problem-solving method, that can solve the task. The mechanisms are specialized -- as described in section 5.1. They define the kinds of domain knowledge that is applicable within each step, and thereby **specify the task concept model.** This frees the developer from creating the task concept model. It is made explicit by the task mechanisms.

Then, a list of the identified mechanisms is passed to another program, called BURN. BURN manages a collection of specialized knowledge acquisition tools that can be used to **extend, operationalize, and maintain the task concept model.** Each of these tools elicit and encode the knowledge needed by one of the identified mechanisms. BURN invokes the knowledge acquisition tools. The tools elicit and encode the knowledge by interacting with the developer. BURN then combines the acquired knowledge with the configured problem-solving method. The result is an application system, a prototype system that automates the application task.

Since SPARK, as well as BURN, use heuristics in configuring the problem-solving method and eliciting the task concept model, there is no guarantee that the application system will solve the task. Thus, the developer uses a debugger, called FIREFIGHTER, to ensure that the application system has the same characteristics than the application task, i.e., the application system consumes the input available to produce the required output. There are several reasons why a mismatch between application systen and task is possible: (1) BURN has not yet elicited enough knowledge, (2) a task mechanism cannot quite accommodate the task requirements, and/or (3) a task mechanism is not appropriate for the application task. FIREFIGHTER assists the developer in determining and resolving the problem. It can be used to acquire more knowledge, change a task mechanism and the associated knowledge acquisition tool, and/or replace a task mechanism and the associated knowledge acquisition tool.

Acknowledgments

The life-cycle for knowledge-based systems is adapted from a model developed by the "Tools" working group of the 3rd Knowledge Acquisition for Knowledge-based Systems Workshop, Banff, Canada, November 1988. The group consisted of Georg Klinker, Marc Linster, Mary Meyer, Enrico Motta, Mark Musen, Stephen Regoczei, and Brian Woodward. The interpretation and conclusions in this paper are those of the author and should not be interpreted as representative for the group. The proposal for an integrated framework for knowledge acquisition presented in section 8 is part of the research program of the Artifical Intelligence Research Group at Digital Equipment Corporation. The group is headed by John McDermott; other researchers involved in the program are Georg Klinker, David Marques, and David Tung. Bill Birmingham, John McDermott, and David Marques provided very useful comments on an earlier version of this paper.

References

[Bachant 88] J. Bachant.
RIME: Preliminary Work Toward a Knowledge-Acquisition Tool.
In *S. Marcus (Ed)., Automating Knowledge Acquisition for Expert Systems.* Kluwer Academic Publishers, Boston, August 1988.

[Barker 89] V. Barker and D. O'Connor.
Expert Systems for Configuration at Digital: XCON and Beyond.
Communications of the ACM 32(3):298-318, 1989.

[Birmingham 88] 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.

[Bobrow 83] D. Bobrow and M. Stefik.
The Loops Manual.
Xerox Corporation, December 1983.

[Boose 84] 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 87] J. Boose and J. Bradshaw.
Expertise Transfer and Complex Problems: Using AQUINAS as a Knowledge Acquisition Workbench for Expert Systems.
International Journal of Man-Machine Studies 26(1):3 - 28, 1987.

[Boose 88] J. Boose.
A Survey of Knowledge Acquisition Techniques and Tools.
In *Proceedings of the 3rd Knowledge Acquisition for Knowledge-based Systems Workshop*. Banff, Canada, 1988.

[Buchanan 78] B. Buchanan and E. Feigenbaum.
DENDRAL and MetaDENDRAL: Their Applications Dimensions.
Artificial Intelligence 11, 1978.

[Chandrasekaran 87]
B. Chandrasekaran.
Towards a Functional Architecture for Intelligence Based on Generic Information Processing Tasks.
In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*. Milano, Italy, 1987.

[Clancey 84] W. Clancey.
Classification problem solving.
In *Proceedings of the 4th National Conference on Artificial Intelligence*. Austin, Texas, 1984.

[Diederich 87] J. Diederich and I. Ruhmann, and M. May.
KRITON: A Knowledge Acquisition Tool for Expert Systems.
International Journal of Man-Machine Studies 26(1):29 - 40, 1987.

[Dietterich 82] T.G. Dietterich .
Learning and Inductive Inference.
In *P.R. Cohen, and E.A. Feigenbaum (Eds.), The Handbook of Artificial Intelligence*. Morgan Kaufmann, Los Altos, CA, 1982.

[Duda 84] Duda, R. and R. Reboh.
AI and Decison Making: The PROSPECTOR Experience.
In *W. Reitman (Ed)., Artificial Intelligence Applications for Business*. Ablex, Norwood, N.Y, 1984.

[Eshelman 87] L. Eshelman and D. Ehret and J. McDermott and M. Tan.
MOLE: A Tenacious Knowledge Acquisition Tool.
International Journal of Man-Machine Studies 26(1):41 - 54, 1987.

[Forgy 81] C.L. Forgy.
OPS5 user's manual.
Technical Report, Carnegie-Mellon University, Department of Computer Science, 1981.

[Friedland 85] P.E. Friedland and Y. Iwasaki.
The Concept and Impementation of Skeletal Plans.
Journal of Automated Reasoning 1(2):161 - 208, 1985.

[Gale 87] W. Gale.
Knowledge Based Knowledge Acquisition for a Statistical Consulting System.
International Journal of Man-Machine Studies 26(1):55 - 64, 1987.

[Gruber 87] T. Gruber and P. Cohen.
Design for Acquisition: Principles of Knowledge System Design to Facilitate Knowledge Acquisition.
International Journal of Man-Machine Studies 26(2):143 - 159, 1987.

[Gruber 88] T. Gruber.
A Method for Acquiring Strategic Knowledge.
In *Proceedings of the 3rd Knowledge Acquisition for Knowledge-based Systems Workshop*. Banff, Canada, 1988.

[Hayes-Roth 83] F. Hayes-Roth, D.A. Waterman, and D.B. Lenat.
Building Expert Systems.
Addison-Wesley, Reading, MA, 1983.

[Kahn 85] G. Kahn and S. Nowlan and J. McDermott.
MORE: an intelligent knowledge acquisition tool.
In *Proceedings of Ninth International Conference on Artificial Intelligence*. Los Angeles, California, 1985.

[Kahn 87] G. Kahn and E. Breaux and R. Joseph, and P. DeKlerk.
An Intelligent Mixed-Initiative Workbench for Knowledge Acquisition.
International Journal of Man-Machine Studies 27(2):167 - 179, 1987.

[Klinker 88a] G. Klinker and Casey Boyd and David Dong and Jon Maiman and John McDermott and and Robert Schnelbach.
Building Expert Systems with KNACK.
In *Proceedings of the 3rd Knowledge Acquisition for Knowledge-based Systems Workshop*. Banff, Canada, November 1988.

[Klinker 88b] 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.

[Kunz 84] Kunz, J., T. Kehler, and M. Williams.
Applications Development Using a Hybrid AI Development System.
AI Magazine 5(3), 1984.

[Laird 87] J.E. Laird and A. Newell and P.S. Rosenbloom.
SOAR: An Architecture for General Intelligence.
Artificial Intelligence 33(1):1 - 64, 1987.

[Lenat 86] D.B. Lenat and M. Prakash, and M. Shepherd.
CYC: using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks.
AI Magazine 6(4):65 - 85, 1986.

[Linster 88] M. Linster.
A Critical Look at KRITON: Moving from First to Second Generation.
In *Proceedings of the 3rd Knowledge Acquisition for Knowlege-Based Systems Workshop*. Banff, Canada, 1988.

[Marcus 88] 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, Boston, August 1988.

[Marques 88] 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, Minnesota, August 1988.

[McDermott 82] J. McDermott.
R1: a rule-based configurer of computer systems.
Artificial Intelligence 19(1), 1982.

[McDermott 88] J. McDermott.
Preliminary Steps Toward a Taxonomy of Problem-Solving Methods.
In *S. Marcus (Ed)., Automating Knowledge Acquisition for Expert Systems*. Kluwer Academic Publishers, Boston, August 1988..

[Motta 88] E. Motta and T. Rajan and M. Eisenstadt.
A Methodology and Tool for Knowledge Acquisition in KEATS-2.
In *Proceedings of the 3rd Knowledge Acquisition for Knowledge-based Systems Workshop*. Banff, Canada, 1988.

[Murray 88] K. Murray and B.W. Porter .
Developing a Tool for Knowledge Integration: Initial Results.
In *Proceedings of the 3rd Knowledge Acquisition for Knowledge-based Systems Workshop*. Banff, Canada, 1988.

[Musen 88] M. Musen.
Generation of Model-Based Knowledge Acquisition Tools for Clinical-Trial Advice Systems.
PhD thesis, Department of Computer Science, Stanford University, 1988.
STAN-CS-88-1194.

[Shaw 87] 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.

[Steele 84] G. Steele.
Common LISP: The Language.
Digital Press, Billerica, Mass., 1984.

[van de Brug 86] A. van de Brug and J. Bachant and J. McDermott.
The Taming of R1.
IEEE Expert 1(3):33 - 38, 1986.

Acquisition, Classification and Formalisation of Software Specification Heuristics

Celso Niskier
Tom Maibaum

Theory and Formal Methods Section
Dept. of Computing
Imperial College of Science and Technology
180 Queen's Gate
London SW7 2AZ

e-mail:
cn@doc.ic.ac.uk
tsem@doc.ic.ac.uk

Abstract

Software specification is seen as a knowledge acquisition activity. A pluralistic model of the software specification acquisition process is proposed, in which multiple viewpoints/formalisms are integrated to enhance our understanding of the required system. In particular, we investigate how this process can be supported by heuristics acquired from well-known software specification methods/notations such as Data Flow Diagrams, Entity Relationship Models and Petri Nets. We suggest the classification of heuristics by method and activity, and show how they can be formalised in Prolog. A practical by-product has been the generation of "expert-assistance" to the integration of methods: PRISMA is a pluralistic knowledge-based system supporting the coherent contruction of a software specification from multiple viewpoints. Examples showing its applications are provided.

Key Words: Software Specification, Views, Heuristics, Knowledge Acquisition

1 Introduction

Many recent research efforts have been reported on the possibility of relating Artificial Intelligence to Software Engineering. Some of them refer, in particular, to the current need for a better understanding of the software development process to allow for a more extensive automation of software production (Balzer, 1985).

AI can be associated with this problem in two ways. On the one hand, it can provide new formalization tools - logic programming, epistemic logics, dialogue logics - to the expression of software engineering concepts. On the other hand, it is possible to draw an analogy between heuristics for problem solving in AI and heuristics used when one is involved with the more general question of solving a problem through a software system. As we shall see, the present work has drawn its inspiration from both approaches.

Software development methods deal with different levels or aspects of the software development process. Software specification acquisition, software design (programming-in-the-large) and program construction (programming-in-the-small) encompass intellectual activities with different characteristics. For each aspect considered there exist methods tested in practice that represent practitioner's current understanding of problem solving through software.

In particular, the area of software specification acquisition has been recognized as one of the most critical in the software development process (Lehman et al, 83; Maibaum, 86). As one moves from an informal knowledge of an application to a formal (or at least "formatted") representation of it, errors are often introduced due to incorrect or ill formed understanding of the desired properties of the system.

Several different methods have been proposed to help bridge the informal/formal gap. It is our firm belief that each method emphasises the use of particular *views* of the problem, and no method is general enough to be used for all classes of problems. Our purpose is, therefore, to investigate conditions under which different methods can be combined to enhance problem-solving ability and to allow for a more comprehensive acquisition of knowledge about software specification.

We have formalized some of these methods by using logic programming (as a metalanguage) to capture their syntactic, semantic and pragmatic (heuristic) aspects. A practical by-product has been the generation of "expert-assistance" for the methods: PRISMA (Portuguese for prism) is a pluralistic knowledge-based system for software specification acquisition (Niskier et al, 1989).

2 The Software Specification Process

Many different analogies can be used to describe the process of software specification acquisition. An interesting one relates specification building to *scientific theory construction*. In Science, relevant concepts are *identified*, a theory is *structured* to describe the phenomenon under consideration and then it is *validated* to see if it matches expectation.

Specification building differs from the scientific paradigm in a very important sense. There is generally no part of the world being modeled. The specification describes an imagined artifact whose realisation will become part of the world. This compounds the difficulty of scientific hypothesising in at least two ways. Firstly, the 'knowledge' of the imagined artifact is much less precisely delineated - no 'world' to act as a check. Further, the very act of realising the artifact may profoundly change the world. Secondly, it is our firm belief that what we can say through a

software specification is heavily dependent on the choice of specification language (or logic) used to express it. Moreover, we believe that, during early steps in the process of software specification acquisition the choice of any specific formalism could severely restrict or bias our ability to prescribe the nature of the intended system (Maibaum, 1986).

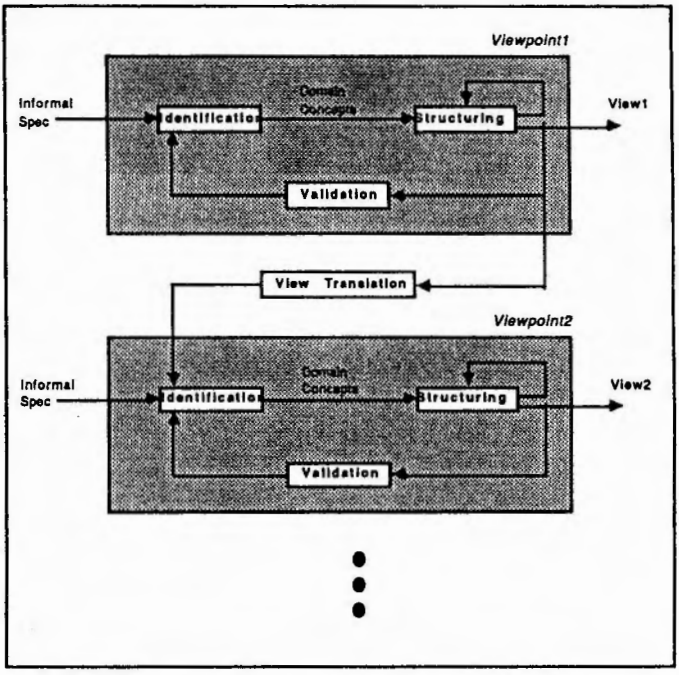


Fig.1 - A Pluralistic Model of Software Specification Acquisition

In our model, the gap between an informal concept of the application and a formal specification of it is bridged by identification, structuring and validation steps. Considering the use of multiple viewpoints/formalisms to enhance the understanding of the system, the complete model, including translations between viewpoints, can be pictured as in Figure 1.

The last step - integration of multiple views into a single formal specification - will not be considered in this paper. However, assuming that earlier steps have guaranteed by construction that the resulting views are mutually consistent, the problem of integrating them into a single formal specification is much simpler, particularly if the semantic and syntactic characterisation of each view is expressed in the same metalanguage (Prolog in the case of PRISMA).

3 Software Specification Heuristics

Many so-called methods are advocated in Software Engineering emphasising desirable properties of the end product, whether that is a specification, design or program. A true method should consist of *rules* for guiding and organizing an activity: the purpose of a method is to guide the practitioner in getting from one point to another. However, although a method should be prescriptive, in that it provides the practitioner with clear instructions about what to do next, whenever there is a large degree of choice, it would be wrong to expect that the software specification process reduces to a recipe or algorithm. No method alone can make the inherently difficult task of understanding a complex domain easy. Rather, a specification method should provide a set of prescriptive **heuristics** (Lenat, 1982; Pearl, 1984) for *focusing attention* on certain aspects at some stages in the construction and analysis of a specification.

The choice of Data Flow Diagrams (DFD) (DeMarco, 1979), Entity Relationship Models (ERM) (Chen, 1976) and Petri Nets (PN) (Peterson, 1981) as test-case methods in our framework was based on some assumptions about their nature. Firstly, each one of them focuses on different aspects of the problem domain, respectively functional, relational and operational views (Mastro, 1985; Yourdon, 1986). This constitutes a satisfactory selection of formalisms for a whole range of problems. Secondly, they represent well-known and widely used methods. The fact that they have been heavily tested in practice adds both to the quantity and quality of the methodological knowledge available. Finally, all three methods make use of graphical formalisms to express knowledge about the system. This uniform characteristic, apart from rendering the approach much more intuitive and appealing - adding the effective possibility of *visualizing* in different ways the system behaviour and structure - favoured the use of a common conceptual basis and an unifying metalanguage in the form of Horn clauses.

According to our previous model, our methodology has been the classification of heuristics by method and activity. Thus, for each test-case method we have acquired from textbooks and interviews with practitioners heuristics for *identification* of concepts, *structuring* and *validation* of the specification, and, for each combination of two methods, heuristics for checking *complementarity* between them.

Identifying concepts is a critical step as once they are identified and used in building the description, the boundaries of the specification are set and difficult to change. This reflects the important rôle of language in the formalisation process which we refered to above. In determining which concepts should appear in the software specification **identification heuristics** are useful. They act as *filters* over a multitude of possible choices from the problem domain, and obviously depend heavily on the underlying formalism. Examples:

- *In a DFD, if a file has no input dataflows then look for the information that is being recovered from it and identify flows indicating when it was stored*

- *In a PN, if an event has no preconditions then look for the state of the system just before the occurence of it and identify relevant conditions characterizing this state*

Structuring heuristics are advice-giving heuristics capturing *past experience* in the application of the method to specific problems. They make use of syntactic properties to characterise unsatisfactory situations in a specification - usually caused by minor inconsistencies or incorrectness - and provide advice on how to overcome these problems. Examples are:

- *In a DFD, if a process has too many inputs and outputs, then refine it into more detailed processes*

- *In an ERM, if an entity has too many attributes, then abstract in into a more general entity*

In our work, a particular example of **validation heuristics** are rules capturing "interesting" properties of specifications in the form of *natural language paraphrases*. A specification may be syntactically correct, but may not have some desired property, or may have some non-desired property, thus indicating a possible semantic mismatch. In this case, validation heuristics suggest how to "read" the specification pinpointing such problems. For example:

- *In a PN, if there is a conflictuous situation then ask whether all preconditions of the corresponding event correctly represent the system state*

- *In an ERM, if there is a one-to-one relationship between two entities, then ask whether instances of one entity interact with just one instance of the other entity*

Complementarity heuristics are the most crucial heuristics in the context of a multiple view specification, as they act as guidelines in assuring the joint consistency of different descriptions. They can be regarded as rules which aid in the construction of a proof of *partial translation correctness* between different views. They indicate ways of verifying that some properties in one view are correctly represented in other views. They represent, therefore, a measure of the overall consistency of the multiple view specification. Few examples:

- *Every process in a DFD should be triggered by the occurrence of at least one event in the corresponding PN*

- *Every event in a PN should represent the change of at least one attribute of an entity in the corresponding ERM*

- *Every entity in an ERM should be created by at least one process in the corresponding DFD*

4 PRISMA: A Pluralistic Knowledge-Based System

To investigate the practical use of the heuristic knowledge previously obtained we have represented such aspects of each method in a unified Prolog-like notation. For each graphical formalism we have identified primitive components, i.e., conceptual units used to describe real world phenomena. Figure 2 ilustrates how each formalism is decomposed into its primitive components.

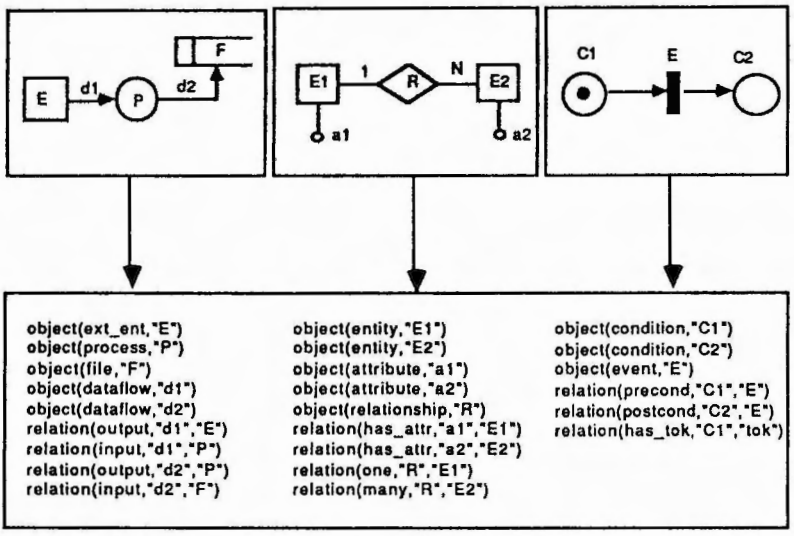


Fig.2 - Decomposition of a multiple view specification into its primitive components

The 'semantical net' treatment given to formalisms, with their consequent formalization in logic, appears to be generic enough to suggest an uniform treatment for a wide class of graphical formalisms. Once primitive components were identified, we have used different combinations of them to describe *syntactic* and *semantic* properties of the view. An example of each can be seen below:

"A process has no input dataflows in a DFD"

```
has_no_inputs(Proc) <-
    object(process,Proc),
    findall(Y,relation(input,Y,Proc),Inputs),
    empty(Inputs).
```

"Two events are in conflict in a PN"

```
conflict(Ev1,Ev2) <-
    object(event,Ev1),
    object(event,Ev2),
    not(equal(Ev1,Ev2)),
    findall(Y,relation(precond,Y,Ev1),Preconds1),
    findall(Y,relation(precond,Y,Ev2),Preconds2),
    marked(Preconds1),
    marked(Preconds2),
    intersection(Preconds1,Preconds2,Common_conds),
    not(empty(Common_conds)).
```

Heuristics - conceptually represented as pairs *<situation,action>* - are formalised in PRISMA as Horn clauses, where situations are characterised by a conjunction of satisfiable queries to an instance of a view. Actions are then specification tasks that need to be performed according to the requirements of the present situation. The following are examples, respectively, of identification, validation and complementarity checking heuristics formalised in Prolog:

"In a DFD, if a process has only output dataflows, then look for the information needed to produce the outputs and indentify relevant input dataflows"

```
context(rule(20),structuring,dfd) <-
    /* if-situations */
    has_outputs(Proc,Outputs),
    has_no_inputs(Proc),
    /* then-actions */
    show_task('Define inputs of process ',Proc),
    show_advice('Look for information or signals needed to
          produce ',Outputs).
```

"Given a one-to-many relationship in an ERM, check if it is correct by generating the following paraphrase: 'an instance of the first entity interacts with many instances of the second one, but only one instance of the second one interacts with the first one'"

```
context(rule(64),validation,erm) <-
    /* if-situation */
    one_to_many(Ent1,Rel,Ent2),
    /* then-action */
    show_paraphrase('An instance of ',Ent1,'interacts
          via relationship',Rel,'with many instances
          of',Ent2),
    show_paraphrase('An instance of ',Ent2,'interacts
          via relationship',Rel,'with one instance
          of',Ent1).
```

"Every entity in a ERM should be associated with at least one file in the corresponding DFD"

```
context(rule(75),change_view(dfd),erm) <-
    /* if-situation */
    object(entity,Ent),
    /* then-action */
    show_comp_check('Check that the entity ',Ent,'is possibly
          associated with a file').
```

The paradigm for the use of PRISMA as a *knowledge-based system* for software specification acquisition has been centered on the idea of an "user-specifier", sitting in front of his/her workstation, trying to build up a structured description of the desired system from informal concepts (e.g. annotations, tables, natural language statements) previously obtained in a

preliminary brainstorming session with the client. For that purpose, different views suggest complementary ways to look at these concepts. The PRISMA environment provides the user with three sets of tools for each view: the *agenda*, the *paraphraser* and the *complementarity checker* (view translation mechanism).

The **agenda** mechanism implements the set of advice-giving heuristics, driven by syntactic rules. At any step, during a view construction, the user-specifier can query, through a tasks window, a list of the remaining and most promising tasks to be performed, with an advice window suggesting how to do each task.

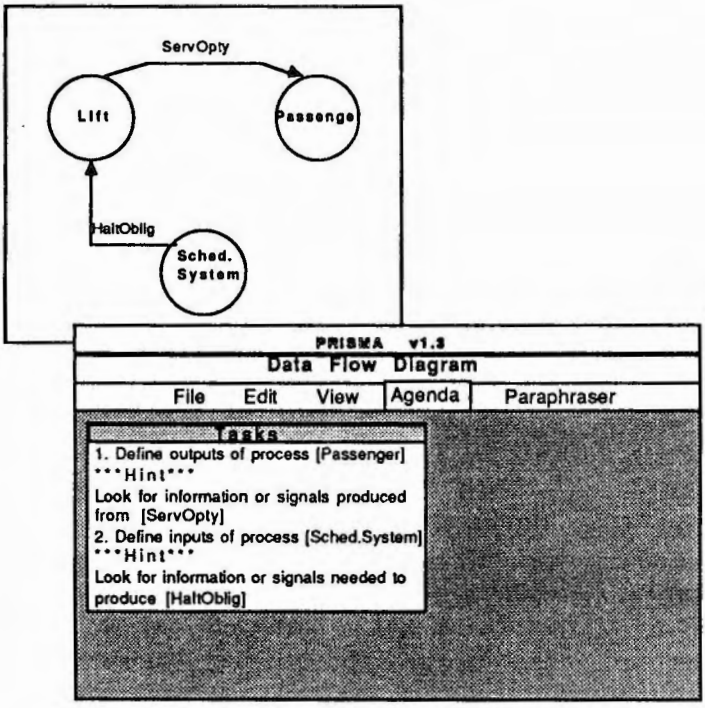


Fig.3 - Data Flow Diagram Agenda

The **paraphraser** makes use of the set of validation heuristics to generate sentences in natural language. A paraphrases window is produced whenever the user wants to scrutinize the current state of a view in order to see if its properties satisfy his original intention.

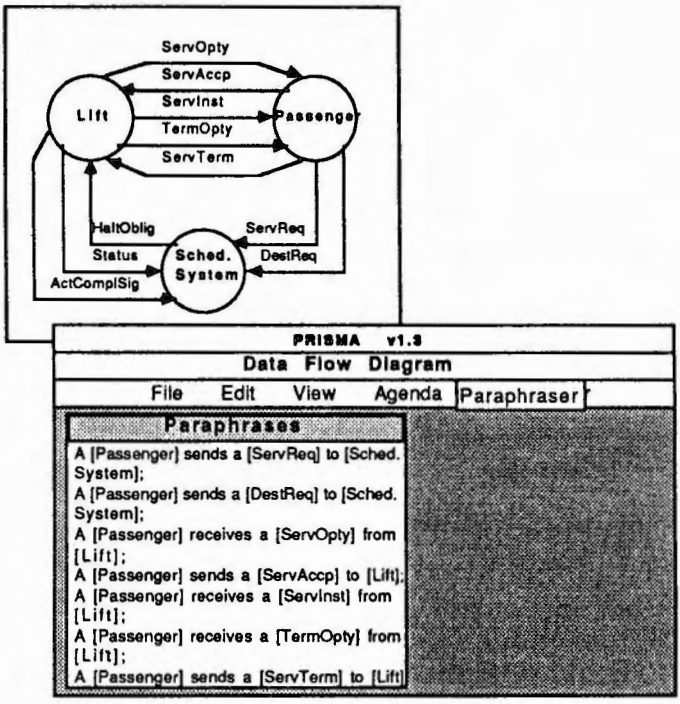


Fig. 4 - Data Flow Diagram Paraphraser

Finally, once he is happy with the possibilities of one view, he/she is allowed to shift to another one. When selecting a new view, the control mechanism automatically switches to the **complementarity checker**, which promptly presents a set of "hints" through a special comp_checks window indicating associations that must be observed between properties of the current view and properties in the new view. This tool can be used in a dual sense: as an aid in constructing a new view from the previous one or as support for checking joint consistency of two complementary views already obtained.

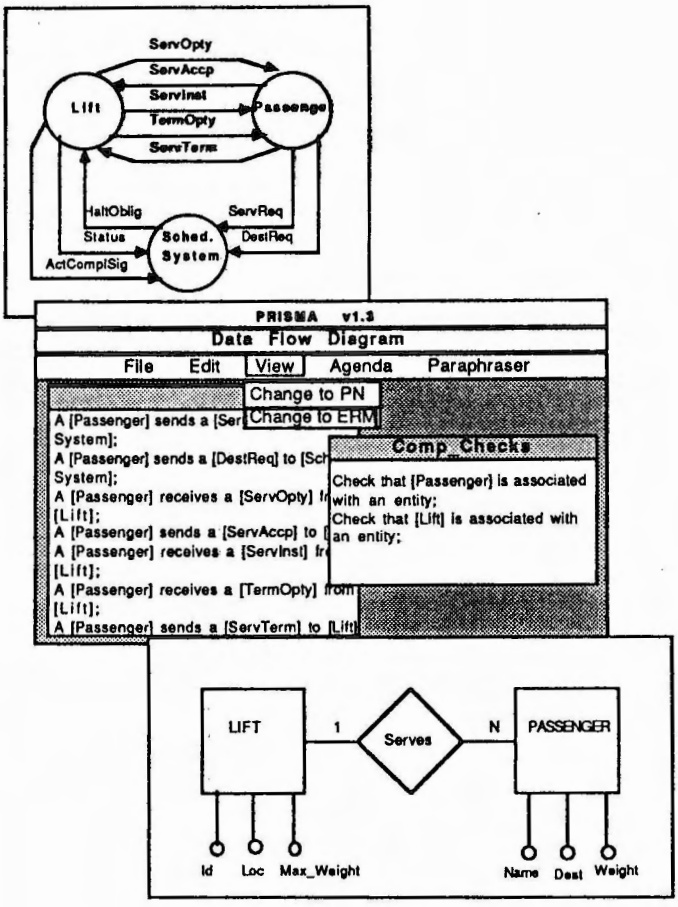


Fig. 5 - Changing from DFD to ERM

5 Conclusions and Further Work

We have shown how software specification heuristics can be classified and formalised according to a pluralistic model of specification acquisition. As a by-product, we have constructed a knowledge-based tool supporting software specification from multiple viewpoints. PRISMA has been tested with medium-sized problems (e.g. an academic information system). The results have been satisfactory, giving us confidence that the combination of multiple viewpoints with effective heuristics provides a good paradigm for software specification acquisition. In fact, we believe that the model can be easily extended to encompass other methods and areas of expertise.

We are currently investigating certain combinations of methods aiming at the extraction from experience of more general heuristics on how to combine particular methods according to problem characteristics.

Acknowledgements

Profs. Carlos Lucena and Daniel Schwabe have been instrumental in inspiring many of our ideas back at PUC in Rio de Janeiro. Among our colleagues at Imperial College Anthony Finkelstein, Hugo Fuks and Martin Sadler contributed with many useful comments to this work. Celso Niskier is supported by the Brazilian National Research Council CNPq, grant no. 20.2518/86-CC.

References

Balzer, R. "A 15 Year Perspective on Automatic Programming", *IEEE Transaction on Software Engineering*, Vol. SE-11, 1985

Chen, P. "The Entity Relationship Model - Toward a Unified View of Data", *ACM - TOD*, Vol.1, no.1, 1976

DeMarco, T. *Structured Analysis and System Specification*, Prentice-Hall, 1979

Finkelstein, A.C.W., Fuks, H., Niskier, C. & Sadler, M. "Constructing a Dialogic Framework for Software Development", *Proc. of the 4th. IEEE International Software Process Workshop*, Devon, England, 1988

Finkelstein, A.C.W. & Potts, C. *Evaluation of Existing Requirements Extraction Strategies*, Alvey FOREST Deliverable Report 1, GEC Research Laboratories, Marconi Research Centre, Great Baddow, Chelmsford, England, 1985

Lehman, M.M., Stenning, V. & Turski, W.M. *Another Look at Software Design Methodology*, Imperial College of Science and Technology, Research Report DoC 83/13, 1983

Lenat, D.B. "The Nature of Heuristics", *Artificial Intelligence* 21(1), 1982

Maibaum, T.S.E. "The Role of Abstraction in Program Development", *Proceedings of the IFIP World Computer Congress*, 1986

Mastro, V.A. "Three Dimensional System Development", *Software Engineering Notes* 10(5), 1985

Niskier, C., Maibaum, T.S.E. & Schwabe, D. "A Look Through PRISMA: Towards Pluralistic Knowledge-Based Environments for Software Specification Acquisition", *Proc. of the 5th International Workshop on Software Specification and Design*, Pittsburgh, USA, 1989

Pearl, J. *Heuristics*, Addison-Wesley, 1984

Peterson, J.L. *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, 1981

Yourdon, E. "What Ever Happened to Structured Analysis ?", *Datamation*, June 1986

Model Directed Inference Structure Acquisition

Erhard KONRAD

TU Berlin
FB20 WBS
Franklinstrasse 28/29
1000 Berlin 10
West Germany

Xuejun TONG*

BULL - Advanced Studies Department
7 rue Ampère
91343 MASSY CEDEX
FRANCE
Email: tong@masbull.gtmy.bull.fr

Abstract

Knowledge can be described at different levels according to its functionality in the problem solving process. We can depict knowledge at the strategic, the task, inference structure, and the conceptual structure levels. Among these levels, the inference structure level is the most domain independent level. It is also the most important in characterizing the problem solving process and in assisting knowledge acquisition. A system called MAC which uses the model directed approach to acquire inference structures is introduced. MAC is a general knowledge acquisition system since inference structures are the most common media among different domains. MAC has a library (called OLD_WORLD) of models of inference structures, represented in clausal form. These models describe different types of problem solving, which are generalized from domains known by knowledge engineers. Facing an unknown domain, MAC interviews domain experts to get a description of the problem solving process in descending order of generality. It then uses graph partition techniques to divid the description, and further more, it goes through a set of meaning preserving operations to construct the inference structure, guided by models of the inference structures known. At the end of the paper, the usefulness and advantage of the MAC, as well as a comparison with other tools are given.

1- Introduction

Knowledge acquisition is an acknowledged bottleneck in KBS development. Techniques have been developed to deal with this bottleneck. Many first generation knowledge acquisition tools have been built[Boose, 1988]. The knowledge acquisition tools fall into two large categories. One is best characterized by the expression "symbol-level knowledge elicitation", that means tools in this category both use theories from psychology (for example: the personal construct theory of Geory Kelly[Kelly, 1959]) and cognitive science and simulate experiences accumulated by knowledge engineers eliciting symbol-level knowledge from either literature or interview or text or protocols[Linster, 1988]. Examples of them are KRITON[Diederich, 1986], ETS[Boose, 1986] and AQUINAS(an extended version of the expertise transfer system ETS) [Boose, Bradshaw, et.al. 1988] and etc. These tools are necessary to obtain basic terminological knowledge as well as their distinguished features. The terminological knowledge acquired by these tools are domain dependent. But they are not suited to acquire structural relations among objects at higher levels of knowledge, or to describe strategic knowledge organizing the problem solving procedure.

Another category is best characterized by the expression "model-directed". Tools in this category are backed up by the methodologies of building expert systems like KADS[Breuker and Wielinga 1985] and by the abstract problem-solving structures accumulated from expert systems built[Clancey, 1985] or by the general shells like ZDEST-2[Tong, He and Yu 1988]. These tools are easier to face the complex real-world, because they know what to expect or look for(which types of concepts and relations are important, how to obtain domain knowledge, etc.) based on the different models they have at hand. "In the use of such tools, the human-computer dialog does not center on the rules, frames, or other symbol structures in the knowledge base; instead the interaction concentrates on the methods by which task specific knowledge is used to arrive at a problem's solution[Musen, 1988]. From these

* This work was done in TU Berlin, when the author was there.

points of view, we believe the model-directed approach is a faster and more practical approach, and it provides the only chance for a general purpose knowledge acquisition tool to achieve generality.

ROGET [Bennet, 1985] is the first tool which adopted model directed approach in knowledge acquisition. It helps the knowledge engineer with the acquisition of the conceptual structure described in a set of domain independent terms, such as advice or result of the system, the evidence or data available to the system and possible inference steps. It contains a set of diagnostic tasks an expert system might perform, such as determine-problems, determine-causes, recommend-actions. These tasks known are used to guide the acquisition of the conceptual structure and to check the consistency of the conceptual structure. A detailed comparison between ROGET and the work reported here will be given at the end of the paper. OPAL [Musen and Fagan, 1987] is also an example of model directed approach. It is a graphical tool that allows cancer specialists to enter new treatment plans into the knowledge base of an expert system called ONCOCIN[Shortliff, et.al. 1981].

What we have built is also a model directed knowledge acquisition tool. It is different from OPAL, FIS[De Jong, 1987], KRIMB[Cox and Blumenthal, 1987] in the sense that it is backed up by the methodology of building expert systems KADS and a working general tool ZDEST; and it emphasizes the acquisition of structural information(inference structures of a domain). Thus our work is on a more abstracted level and is more suitable to be applied to a larger number of domains.

According to KADS and ZDEST-2 , knowledge can be described at four different levels based on epistemological distinctions: they contain different types of knowledge. [Breuker and Wielinga 1985] gives a very clear description of the division of the levels:

> *We distinguish between static knowledge describing concepts and relations, knowledge representing elementary tasks, and strategic knowledge. Each of these categories of knowledge is described at a separate level. The separation reflects different ways in which the knowledge can be viewed and used. The first layer contains the static knowledge of the domain: domain concepts, relations and complex structures, such as models of processes or devices. The second layer is the inference layer. In this layer we describe what inferences can be made on the basis of the knowledge in the static layer. Two types of entities are represented at the inference layer: meta-classes and knowledge sources. Meta-classes describe the role that domain concepts can play in a reasoning process. .... Knowledge sources describe what types of inferences can be made on the basis of the relations in the domain layer.*
>
> *The third layer is the task layer. At this level the basic objects are goals and tasks. Tasks are ways in which knowledge sources can be combined to achieve a particular goal. The fourth layer is the strategic layer in which knowledge resides which allows a system to make plans. The four layers are schematically presented in Figure 1.*

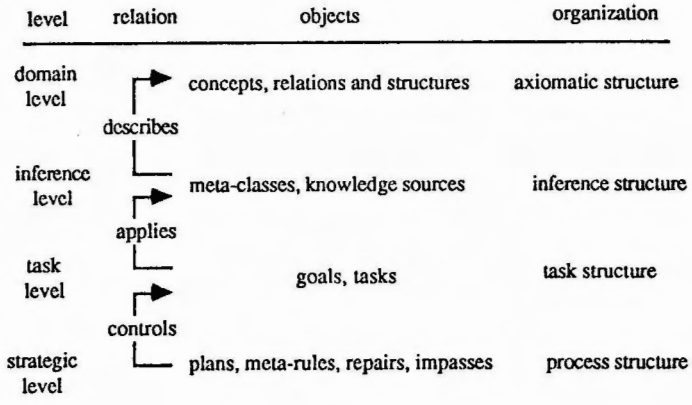


Figure 1. Layers of description of knowledge (from the same paper)

Among these levels, strategic level and task level emphasize control among problem solving components, the knowledge stored at these two levels are domain specific. The inference level(in this paper, we call it inference structure level) describes how to use knowledge in problem solving processes, indicating what inferences can be made in a domain and implicitly what inferences can not be made(The idea of inference level is also explicated in [Clancey, 1985]). The domain level (in this paper, we call it conceptual structure level) describes concepts and their relations, it stores the most domain specific knowledge. Since inference structure is the major concept throughout this paper, we will elaborate its meaning, its functions in problem solving and its roles in knowledge acquisition.

The ***inference structure*** of a domain describes what inferences can be made based on the conceptual structure knowledge of that domain. Two types of entities are represented at this level: *meta-classes* and *knowledge sources*. *Meta-classes* describe the roles that domain concepts can play in problem solving process: How they are to be used and their relative status during the various stages of a problem solving process. For example, a domain concept *infection* can play the role of *finding* in a consultation process or the role of *hypothesis*. *Knowledge sources* describe what types of inferences can be made on the basis of the relations in the conceptual structure(detailed description see [Wielinga and Breuker 1986]). The problem solving process of a domain is characterized by a set of directed inferences and is represented by the ***inference structure***. Tasks are built on the basis of the ***inference structure***, and they impose control knowledge vis-a-vis the division and evolution of the problem solving process on the ***inference structure***. As such, the inference structure is of prime importance in describing domain knowledge and the problem solving process. Moreover, it is the most domain independent level which can be abstracted and shared by different domains on the basis of our assumption 1:

> "Although knowledge may exist in many forms according to the range of problems, we assume that at some high level knowledge can be classified. Domains may contain different concepts, but they may share ways of using knowledge at some higher level of abstraction. This is due to the similarity not only of their problems and solutions, but also of their knowledge structures."

By accepting assumption 1, because of its domain independence, inference structures will play the following roles in knowledge acquisition:

(1) As the model of knowledge structure known in order to guide the acquisition process.

(2) As a media to understand and characterize the problem solving nature of a new domain.

(3) As the subject of further specialization processes.

(4) Guiding the acquisition process by discovering missing components.

And it also plays one particular important role in our research: the inference structure acquired can be used directly by expert system tool ZDEST-2 to form an expert system [Kabach, Tong and Voss 1988].

Because of the inference structure's importance in both problem solving and knowledge acquisition, a system called MAC(Machine Acquisition Component) is built by us to acquire inference structures in a general way. MAC has a library of inference structures represented in clausal form. It interviews domain experts to get a description of the problem solving process in descending generality order. And guided by the inference structures known, it goes through a set of meaning preserving operations to construct an inference structure of the domain. As the result of the construction, MAC "understands" the problem solving process of the domain in the sense that it can use the inference structures already known to reconstruct the inference structure of the domain. And, with the help of the features of the inference structures already known, MAC can tell the features of the new structure and can use the features to guide further acquisition of knowledge and building of expert systems. The following sections give a more detailed description of MAC.

2- Model Representation and the Library

Inference structures are composed of ***knowledge sources*** and *meta-classes*. *Knowledge sources* describe the inferences performed. *Meta-classes* specify the concept types which are inputs of the

inference process defined by the *knowledge source* and results of inference. The relations in the inference structure can be viewed as dependency between different inference operations. The inference structure describes what inference can be made, but it does not specify in what order inference operations are executed. It is a static structure that does not specify a strategy explicitly. For instance, the inference structure of heuristic classification can be described in a graph shown in Figure 2(where rectangle indicates *meta-class* and oval indicates *knowledge source*).

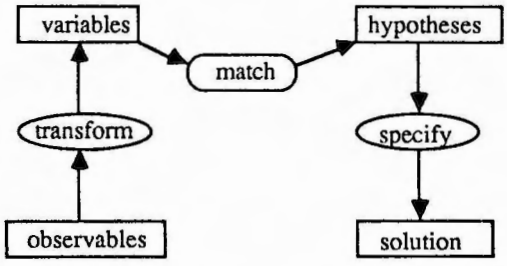
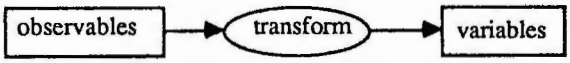


Figure 2. Inference structure of heuristic classification

Formalisms in clausal form are used to describe a series of inference operations. Individual inference operations are defined by atomic formula. The input and output of inference operations are defined as arguments of predicates; and the function of the inference operation is defined by predicate symbol. We give, an example of inference operation:



can be described by an atomic formula: _TRANSFORM(<observables> <*variables*>). And the inference structure of heuristic classification can be described in clausal form(the convention of writing is based on [Kowalski, 1979], while words in italic form indicate outputs of operation):

{ _TRANSFORM(<observables> <*variables*>),
_MATCH(<variables> <*hypotheses*>),
_SPECIFY(<hypotheses> <*solution*>) }

MAC has a library called OLD_WORLD, it has a collection of inference structure models represented in clausal form. These models describe different types of problem solving, they are generalized from domains which are known by KE. They are inference structures describing the problem solving process of generic tasks. For instance, there is a taxonomy of methods as shown in Figure 3 to solve diagnostic problems[Breuker, et.al 1987]:

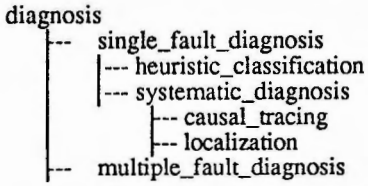


Figure 3. Taxonomy of problem solving methods for diagnosis

OLD_WORLD has all the inference structures for these methods. Besides, inference structures can be defined according to their frequency of occurrence in the problem solving process. OLD_WORLD also stores the definition of predicate symbols, and of *meta-classes*. For example, the predicate _TRANSFORM is defined as:

_TRANSFORM(<observables> <*variables*>)
<---- _INPUT (<observables>),
_OUTPUT(<variables>),
_MEANING(meaning(<transform>))

and *meta-class* <x> is defined as:

_META-CLASS(<x>) <---- _MEANING (meaning(<x>))

where "meaning" is a function to get the natural language definition of the symbol. The definition of a predicate symbol specifies the input and output constraints of the operation and also the natural language expression describing the operation indicated by the predicate symbol. The definition of *meta-class* gives a description of its domain and its meaning in natural language expression.

3- Acquisition of Inference Structures

3.1-. Interviewing Experts

With the existence of the OLD_WORLD, knowledge acquisition is easier to carry out. Firstly, domain experts are interviewed by the KE, and they are taught how to represent natural language sentences in equivalent predicate clausal forms by taking verbs as predicate symbols. For instance, the sentence *"Mushroom classification is a process of matching and distinguishing"* is represented as:

_MUSHROOM-CLASSIFICATION
<---- _MATCH, _DISTINGUISH

And we have assumption 2:

"Domain experts are able to give a description of problem solving evolution in descending order of generality."

By descending order of generality, we mean that experts could give a description first at a very abstracted level without going into any technical detail. And then, step by step, experts will go into more technical details on requirement. This assumption is easy to accept, because we all have the experience that when we are curious about a field we are not familiar with, the experts are sure to inform us by descending order of generality even if we do not know the actual technical process.

During the interviewing process, experts need not consider whether the symbols used by them are consistent with the OLD_WORLD, the meaning classification process will be guided by MAC.

3.2- Meaning Classification

After receiving a general description of the problem solving process, MAC will do the following:

(1) Use the definition of constant symbols, variable symbols and predicate symbols stored in the OLD_WORLD to find out symbols which are referred to by experts and are consistent with the OLD_WORLD. When different vocabulary is used by experts as an alternative reference of one symbol which is known by OLD_WORLD, MAC will prompt the meaning expression of the symbol so as to ask experts to confirm, and at the same time, stores the new expression as another meaning specification of the symbol.

(2) If a predicate symbol is unknown by OLD_WORLD, then the interview process which is at a more specific level is started:

  (a) If the predicate symbol describes a primitive operation, then the experts are asked to give a definition of it.

  (b) If the operation specified by the predicate symbol can be split into more concrete operations, then the experts are asked to give a description of these operations in clausal form.

(3) If a variable symbol or a constant symbol is unknown, then the experts are asked to define it.

(4) Repeatedly doing (1), (2) and (3) until all the predicate symbols, constant symbols and variable symbols have the same meaning as those in OLD_WORLD.

3.3- Inference Structure Recognition

All the inference structure models in OLD_WORLD are frequently occurred grouping of operations, they are generalized from specific domains. But for construction of a realistic inference structure of a domain, usually one inference structure model is not enough, in most cases, a set of operations must be done to compose a complete inference structure for the domain, the operations are:

(1) UNION: several inference models connected sequentially to describe the inference structure.

(2) AND: several models connected parallel by sharing common input symbols.

(3) REFINE: when a model in OLD_WORLD is generally suitable but at some point different, then the model must go through modification in order to be tailored to the specific application. Typical operations of REFINE are *term-substitution, grouping-substitution, missing-part-detection* and *expansion.*

term-substitution: If the domain of real input or output of the operation is a subset of the domain of the input or output specified by the input or output constraints of the operation in the model, then *substitution* is done.

grouping-substitution: A series of sequentially connected operations in the model is substituted by sequentially connected operations peculiar to the domain.

missing-part-detection: If a model implicitly occurs in the inference structure, but there are some parts missing, then MAC will ask the experts to provide the missing parts.

expansion: The final output of the inference structure model may go through some additional operations, because of the particular domain, these operations can be attached to the selected model.

3.4- An Example of Mushroom Classification

Given a pre-defined taxonomy of mushroom classes, mushroom classification is to find out to which class a particular mushroom belongs. The taxonomy is obtained from books. There are many attributes which are used to classify mushrooms, some of their values are easy to obtain, but some are not, because of danger or expensive testing. The task of mushroom classification is to determine the class of mushroom by using proper attributes at appropriate times. The problem is mainly a heuristic classification problem(Muller 1986), but it is not so simple because of the complexity in choosing attributes. Stated in technical terms, it embodies the following procedure:

Some direct observables give rise to a set of first hypotheses, which is reduced to a single hypothesis by further, more expensive observables. For security reasons, this hypothesis is once more differentiated with confusing mushrooms.

We now show the acquisition of an inference structure of mushroom classification. **This is taken directly from executing MAC on VAX in Franz Lisp.**

Firstly, the interview process(words in italic forms are presented by domain experts):

>>>>MAC now starts the acquisition of inference structure! <<<<

>>Please give me a general description in clausal form what is to be done! (Sentences after ">>" are prompted by the system.)
** *_MUSHROOM-CLASSIFICATION <--- _MATCH, _DISTINGUISH*
(Sentences after "**" are presented by the experts.)

>>We have a _MATCH defined as:
_MATCH(<variables> *<hypotheses>*)
<--- _INPUT(<variables>),
_OUTPUT(<hypotheses>),
_MEANING(meaning (<match>))

with meaning of "Abstracted data are matched heuristically by direct, non-hierarchical association with some concept in another classification hierarchy."

Is it the same as yours?
** *Yes*

>> And the meta-classes are also the same ?
** *No. _MATCH (<direct-observables> <hypotheses>)*

>> Good.
What is the meaning of meta-class "direct-observables" ?
** *direct-observables are variables which are easier to obtain.*

>> Good. Is there any operation to obtain the "direct-observables" ?
** *Yes.*
_SELECT-DIRECT-OBSERVABLES (<observables> <direct-observables>)

>> Sorry, I do not know the predicate "select-direct-observables", could you give me a description of its function ?
** *SELECT-DIRECT-OBSERVABLES is an operation to get direct-observables from a set of observables. The observables are classified according to certain attributes. The domain knowledge refers to the names of attributes and the domain of observables.*

>> Good.

The result of the interview process and meaning classification is an inference structure constructed by MAC which is shown in Figure 4. The representation in clausal form is:

{ _SELECT-DIRECT-OBSERVABLES(<observables> *<direct-observables>*),
_MATCH(<direct-observables> *<hypotheses>*),
_SELECT-EXPENSIVE-OBSERVABLES(<observables> <hypotheses>
<expensive-observables>),
_ESTABLISH(<expensive-observables> <hypotheses> *<hypothesis>*),
_CONFOUND-WITH(<hypothesis> *<differential-hypotheses>*),
_SELECT-DIFFERENTIAL-OBSERVABLES(<observables>
<differential-hypotheses>
<differential-observables>)
_DIFFERENTIATE(<differential-observables>
<differential-hypotheses>
<solution>) }

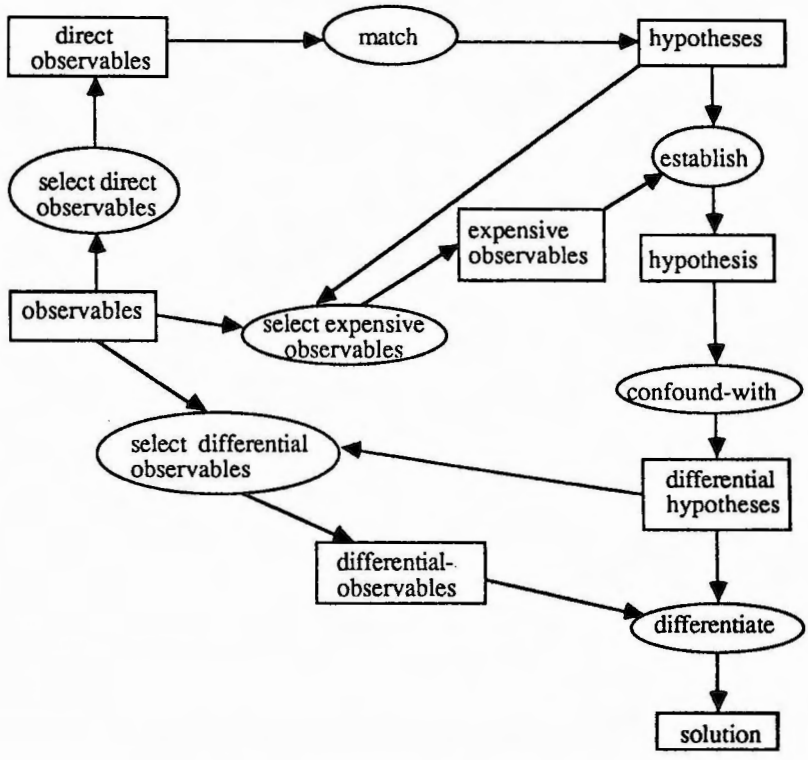


Figure 4. Inference structure of mushroom classification

Transformations done in the inference structure recognition step are:

variables *substituted by* <---------- direct-observables,

_ABSTRACT(<observables> *<direct-observables>*) *substituted by* <----------
_SELECT-DIRECT-OBSERVABLES(<observables> *<direct-observables>*),

and _SPECIFY(<hypotheses> *<solution>*) *substituted by* <----------

{ _SELECT-EXPENSIVE-OBSERVABLES(<observables> <hypotheses>
<expensive-observables>),
_ESTABLISH(<expensive-observables> <hypotheses> *<hypothesis>*),
_CONFOUND-WITH(<hypothesis> *<differential-hypotheses>*),
_SELECT-DIFFERENTIAL-OBSERVABLES(<observables>
<differential-hypotheses>
<differential-observables>)
_DIFFERENTIATE(<differential-observables>
<differential-hypotheses>
<solution>)
}

The result of the inference structure recognition step is a refined inference structure of heuristic classification which is particularly oriented to the mushroom classification domain.

4- Result Discussion

Inference structures specify the expert's inference process with all domain dependent peculiarities being abstracted away. They explicitly distinguish the different types of knowledge being needed to solve the problem. Therefore, they guide categorizing the amorphous mass of elicited knowledge. Additionally, they help to recognize missing knowledge(which is required in the model, but not yet elicited in expert's verbal data). Our work shows an easy way to acquire the functional description of the problem solving process and to construct inference structures for different domains. By choosing clausal form as the formalism for representing verbal knowledge, the functional description provided by domain experts is less ambiguous, and the difficulty of dealing with natural language is partially avoided. In addition, the entire knowledge acquisition process is meaning preserving. That is, the result of acquisition is consistent with the description of experts. This can be shown by the inference procedures of clausal form.

The acquisition of inference structures is very useful and important to the building of the expert systems. The acquired inference structure can assist the acquisition of other domain specific knowledge such as concepts and relations among concepts, because the roles of concepts and relations are already specified by the inference structure. Inference structures can also assist the characterization of tasks, sub-tasks, and strategies, because from one point of view, the construction of tasks and sub-tasks is a division of inference structure. It is useful in fast prototyping of the domain oriented expert systems, because the inference structures of the domains can be reconstructed by the inference structure models in the OLD_WORLD, which have programs to be used directly. In addition, it can be used in analyzing the knowledge reliance for the sake of parallel processing.

The advantage includes the following:

(1) Inference structures describe the structural information of knowledge, they have generality. On the other hand, because of its domain independence, the usage of models is much greater, and the models are more useful in assisting the acquisition of knowledge for other domains.

(2) With the help of the model, the most important information will be picked up. This is of much advantage in comparison with case-based methods.

(3) The result is reliable and unique, it will not be influenced by individual cases or examples.

(4) The result can be used by ZDEST-2 expert system tool which supports the division of knowledge into similar four levels in order to build expert systems.

Experiments using MAC have been carried out. The inference structure of the mushroom classification problem is actually acquired by MAC[Tong, 1989]. However, MAC has not been used by domain experts directly yet, problems such as recognizing predicates which are the same in their functions, and meta-classes which bear the same meaning have not been fully resolved. There are also problems of how to deal with refined parts of the known inference structures so as to produce an complete operational system for the domain.

5- Comparison With Other Tools

Tools which fall in the model directed category are many. But the models used by different tools are quite different. Among these tools ROGET seems to be the most closest to MAC. But looking as the details of the two systems, the differences are many. The key ideas of the ROGET can be described as:

(1) It has a predetermined set of the expert system structures of the MYCIN style called tasks.

(2) Each task has a predetermined set of expectations about the categories of the concepts. The expectation forms a skeletal terminology framework of the domain.

(3) When a task ROGET knows is selected as a model, the skeletal terminology framework is inherited. And domain experts are guided to fill in the instances of the categories so as to get the conceptual structure of the new domain.

The weakness of the ROGET is:

(1) The task set is predetermined, and can not be easily expended.

(2) EMYCIN itself has a limited area of application. So based on the EMYCIN, ROGET will be limited to the narrow area. ROGET is domain dependent.

(3) The conceptual structure of ROGET is dependent on the tasks. The operational model of the task determines the usage of the categories and their instances. When the conceptual structure is refined or expended, another problem may arise: the task may be no more workable.

(4) The categories are expendable, but without guidance.

On the other side, MAC is more general. The categories in ROGET are similar to the meta-classes in MAC. But obviously, MAC accumulates its meta-classes through many ways: either the meta-classes in the OLD-WORLD, or the meta-classes acquired through relational description (Because of the length of the paper, the acquisition of meta-classes is not talked about here.), or the meta-classes recognized in the inference structure acquisition. The most important difference is that MAC allows arbitrary description of the inference structures. The inference structures known by MAC are used to make up the complete domain inference structure. This feature gives MAC much more generality. Besides, the inference structure gives a functional description of the domain problem solving process, it is not an operational model of an expert system, so it is more practical to use. The results provided by MAC will give both domain experts and knowledge engineers better understanding of the domain and the global view about how to organize the problem solving structure in constructing a real expert system.

Compared with OPAL: OPAL has a task-model of expert system ONCOCIN. The task model reflects recurring domain tasks to be performed. Detailed knowledge then is filled in through interview with the tool. The user is released from the work of modelling of his own domain. Though MAC has the OLD-WORLD to store the inference structures taken from known systems, these structures are not used directly to reproduce similar expert systems, instead they are used to compose the inference structures for the domains MAC is unfamiliar with.

PROTEGE[Musen, 1988] is a tool to edit task models similar to the model used by OPAL. "By filling out graphical forms on a workstation screen, knowledge engineers and domain experts can use PROTEGE to define a set of knowledge-level terms and relationships that can model a particular set of application tasks". But the basic model recurred is still the model of ONCOCIN. It did not touch the domains which differ from ONCOCIN on the nature, while MAC is able to attack different domains.

As far as we know, MAC is the only one which is used to acquire inference structures, provided that the inference structure is defined as above.

Acknowledgements

Thanks to all the people who provide comments during the formation of this paper, especially Jianzhong Li and Michel Tueni. Thanks to all the people in UNIBASE group of Technische Universitat Berlin, who provide machine for implementing MAC. And professor Zhijun He is thanked for his encouragement and suggestion.

References

Bennet, J.S. (1985), ROGET: a knowledge based system for acquiring the conceptual structure of a diagnostic expert system. Journal of Automated Reasoning. Vol. 1.

Boose, J.H., (1986) Expertise transfer for expert system design, Elsevier, Amsterdam.

Boose,J.H., Bradshaw, J.M., etc., (1988) Recent progress in AQUINAS: a knowledge acquisition workbench, Proceedings of the second European Workshop on Knowledge Acquisition, Bonn, Germany.

Boose,J.H., (1988), A research framework for knowledge acquisition techniques and tools, Proceedings of the second European Workshop on Knowledge Acquisition, Bonn, Germany.

Breuker, J.A. and Wielinga, B.J. (1985) KADS: structured knowledge acquisition for expert systems. Proceedings of fifth international workshop on expert systems and their applications. Avignon, France.

Breuker, J.A. and Wielinga, B.J. et.al. (1987) "Model-driven knowledge acquisition:interpretation models", Deliverable Task A1, Esprit Project 1098, Memo 87, VF Project Knowledge Acquisition in Formal Domains.

Clancey, W.J. (1985) Heuristic classification, AI 27:289-350.

Chandrasekaran, B. (1986) Generic tasks in knowledge based reasoning: high-level building blocks for expert system design. IEEE Expert, 1:23-30.

Cox, L.A., Blumenthal, R. (1987) KRIMB: an intelligent knowledge acquisition and representation program for interactive model building. Proceedings of the first European Workshop on Knowledge Acquisition for Knowledge Based Systems, Reading University.

De Jong, K. (1987) Knowledge acquisition for fault isolation expert systems. International Journal on Man-Machine Studies, Vol. 27, No. 2.

Diederich, J., (1986) KRITON: a knowledge acquisition tool for expert systems, in the proceedings of the knowledge acquisition for knowledge based systems workshop, Banff, Canada.

Kabach, W., Tong, X. and Voss, A. (1988) Filling up knowledge acquisition gap: from KADS model of expertise to ZDEST-2 expert system. Proceedings of Second European Workshop on Knowledge Acquisition. Bonn.

Kelly, G.A. (1955) The psychology of personal constructs, Norton, NewYork.

Kietz, J.U. (1988) Incremental and reversible acquisition of taxonomies, Proceedings of the second European Workshop on Knowledge Acquisition, Bonn, Germany.

Kowalski, R. (1979) Logic for problem solving. Elsevier North Holland, Inc.

Linster, M. (1988) A critical look at KRITON, Working papers of GMD No. 337.

Muller, B. (1986) Lehrmaterialien BABYLON: Die Beispielwissensbasen zur Pilzbestimmung, Arbeitspapiere der GMD Nr. 221.

Musen, M.A., Fagan, L.M., et.al. (1987) Use of a domain model to drive an interactive knowledge editing tool. International Journal of Man-Machine Studies, Vol. 26, No. 1.

Musen, M.A. (1988) Conceptual Models of interactive knowledge acquisition tools, Proceedings of the second European Workshop on Knowledge Acquisition, Bonn, Germany.

Shortliffe, E.H., Scott, A.C., Bischoff, M.B., van Melle, W., and Jacobs, C.D. (1981) ONCOCIN: an expert system for oncology protocol management, proceedings of the IJCAI-81, p876-891.

Tong, X., He, Z. and Yu, R. (1988) A survey of expert system tool ZDEST-2. Proceedings of eighth European conference on artificial intelligence.

Wielinga, B.J. and Breuker, J.A. (1986) Models of Expertise, proceedings of the 7th European Conference on Artificial Intelligence, p306-318.

EKAW-89

Knowledge Representation for Acquisition and Performance

Integrating Task Demands and Problem-Solving Methods to Develop Useful Taxonomies for Knowledge Acquisition

Brian Woodward
Knowledge Science Institute
Department of Computer Science
The University of Calgary
2500 University Dr. NW
Calgary, Alberta, CANADA
T2N 1N4

Abstract

In order to build useful and meaningful knowledge acquisition tools it is necessary to identify the task demands and the problem-solving methods used to address the problem. Once these two aspects are identified, the types of knowledge needed are determined and elicitation methods are selected or developed to acquire the necessary knowledge. Few attempts have been successful in developing a mapping from task demands to problem-solving methods. Identifying task demands assumes that there are specific observable, objective features of the task situation which can be determined without the involvement of the problem-solver. Determining the method(s) used by the problem-solver to address the task assumes a subjective, constructivist approach for meeting the task demands. Taxonomies or typologies exist for objective tasks and for problem-solving methods but few are useful for building knowledge acquisition tools. This paper reviews a variety of classification systems for both tasks and for problem-solving methods, postulates the 'task-method' as a concept for integration, and develops a set of five criteria for developing task-methods which allows the development of a useful taxonomy for knowledge acquisition. The outcomes of three major methodologies (Rapid Prototyping, Generic Structures, and Knowledge Analysis) of knowledge acquisition are compared against these criteria. Future work includes the construction of a methodology for developing task-methods and a taxonomy of types.

Introduction

At the third Banff Workshop on Knowledge Acquisition for Knowledge-Based Systems, the discussion group on knowledge acquisition tools [1] specified objectives for the future of knowledge acquisition tools. Three of these pertain to the present study:

1. to build tools and to develop procedures that are reflective of explicit, well-validated principles;

1. G. Klinker, M. Linster, M. Meyer, E. Motta, M. Musen, S. Regoczei, B. Woodward

2. to develop methodologies that allow people to translate their mental models into examinable, communicable, notations; and
3. to develop terms and relationships that can provide a vocabulary for describing problem-solving behaviours.

A recurrent theme underlying these objectives was the necessity for a taxonomy or classification strategy of application task demands which provide explicit and direct links to appropriate methodologies or procedures for meeting the task demands. Task modelling emerged as an important research focus for knowledge acquisition tool developers.

Roadblocks to effective task model and procedures specification were also identified:

1. the complexity of real-world tasks frustrates a complete modelling of the task;
2. poorly developed languages exist for describing task features;
3. there is no vocabulary for mapping a task onto a problem-solving method (the method is not evident in initial linguistic analysis nor can it be easily articulated by experts); and
4. different knowledge acquisition tools give incompatible views on the same knowledge.

There exist, however, good reasons for the presence of these roadblocks. The differentiation made between task types and problem solving methods in knowledge acquisition (Boose, 1988) may falsely represent the situation. Defining the task demands separately from the problem solving methods used to solve them may indicate a cause for some of the roadblocks. One of the results is that we are left to identify 'matches' or correspondences between the two separate entities (task and method). This perspective assumes that the elements of real-world or external situations can be categorized without the inclusion of the problem solver - the one who addresses the task.

Rational/Objectivist vs Constructivist Approach to Problem-Solving

A rational/objectivist approach defines a problem as the observable gap between a given goal or predefined standards of performance and the present state of affairs (Nystedt, 1983). Within this framework, the 'task' is an objectively defined set of situational demands which must be met or addressed in order to close the gap. Assumptions are that the gap can be objectified, that the problem-solving process addressing the task can be identified and agreed upon and that the problem can be decomposed into sub-problems. Diagnostic tasks in one domain should by the same as diagnostic tasks in other domains. In comparison, the constructivist approach defines a problem as an artifact of cognitive processes (Nystedt, 1983) and problem-solving is defined as problem-formulation combined with the steps taken to arrive at cognitive closure (eg. satisfaction, reduction of dissonance). problems may be sub-divided but more likely they are formulated and reformulated. This approach assumes that the goal may be shared with others but there may be numerous individual methods of reaching the goal. Different experts in the same domain may use different problem-solving methods to solve the same problems.

Both approaches assume that tasks and methods may be classified based on various qualities and characteristics. This is useful for knowledge acquisition. If knowledge engineers can recognize the 'task characteristics' and the 'problem-solving method' of the expert and match the two, then knowledge base development takes on a more methodical, less ideosyncratic flavour (Welbank, 1987). The salient types of knowledge can be identified and algorithms for the processing of the knowledge constructed or selected. However, there does not yet exist a fully functional taxonomy of either task characteristics or problem-solving methods which can assist knowledge engineers throughout the knowledge acquisition process. Also, there is no useful scheme for mapping specific task demands with problem-solving processes.

A Representative Review of Current Taxonomies and Typologies

This section presents representative samples of classification schemes from three areas of research: artificial intelligence and knowledge acquisition, cognitive science and philosophy. The purpose here is to identify and evaluate the various types of classification schemes and to determine which may be useful for developing effective and meaningful classification schemes for knowledge acquisition.

1. Artificial Intelligence and Knowledge Acquisition

One attempt at a typology of 'problem types' in AI was that of Hayes-Roth et al (1983). These authors identified the following types:

-interpretation	-prediction	-diagnosis
-design	-planning	-monitoring
-debugging	-repair	-instruction
-control		

This scheme was adapted by Kitto and Boose (1987) and Boose (1988) and combined with problem-solving strategies in an attempt to establish a format for the selection of knowledge acquisition tools on the basis of 'application characteristics'. They were able to map two major problem-solving methods (Clancey, 1985) onto 'generic application task categories' by means of automated knowledge acquisition tools. Figure 1. reproduces the scheme.

Figure 2. below represents another attempt to produce a partial taxonomy of problem-solving types. McDermott (1988) has identified a number of 'role-limiting methods' which are easily identified problem-solving methods. He notes that this is a partial taxonomy only but it has inherent in its method the identification of a repetitive cycle of procedures which identifies the required knowledge and processes it.

All of these AI classification schemes, however, suffer from common difficulties. Kidd (1987) points out the problems with this approach to developing classification schemes:

"The utility, and indeed validity, of these categories is questionable for three reasons: (1) What appears to be an inherent property of a real-world problem may rather be a property of how AI programs currently solve that problem. (2) There is evidence that the same task in two different domains may actually involve different kinds of problem, e.g., medical versus electronic diagnosis (Clancey, 1985). (3) Any one of the task categories listed above may, in practice, incorporate a subset of other tasks; e.g. diagnosis may include elements of planning or prediction." (p. 6)

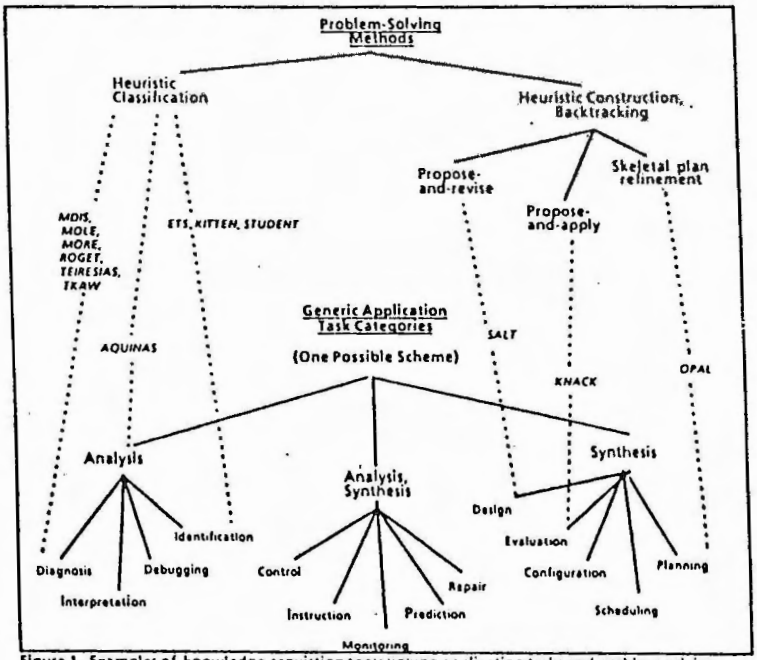


Figure 1. Examples of knowledge acquisition tools linking application tasks and problem-solving methods.

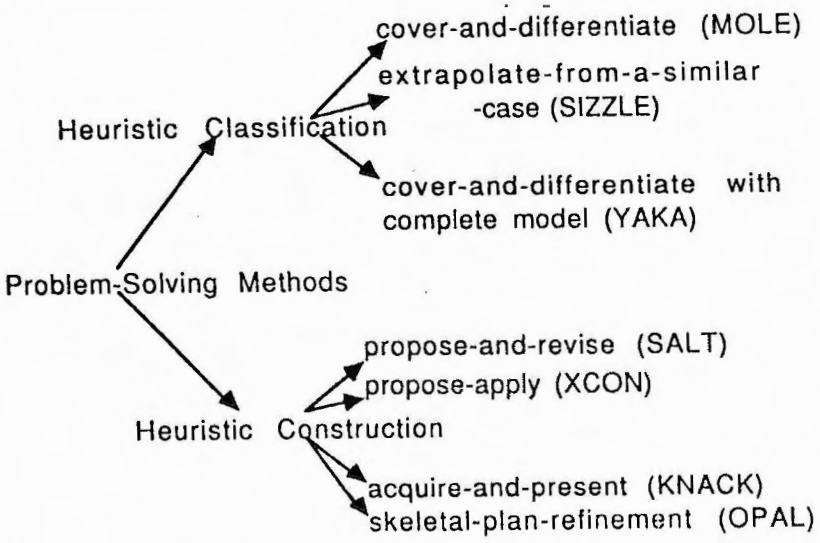


Figure 2 Role-Limiting Methods (McDermott, 1988)

Although the approach taken by Chandrasekaran (1988) assumes some of the same criticisms, he makes an attempt to develop a series of 'generic tasks' which try to emulate how a human problem-solver would address a given set of task demands. The following list outlines some of the generic tasks which have been developed:

-classification -abduction
-hypothesis matching -abductive assembly
-hierarchical classification -concept matching
-plan selection -date retrieval and abstraction

There seems to be an attempt to identify the modules of problem-solving needed to solve certain tasks. Combining these independent modules is the method by which the problem-solving system addresses the task demands. There is some definite confusion here, however, because his 'generic tasks' appear to be more like problem-solving methods. They appear more descriptive of the problem-solver than the objective task.

2. Cognitive Science

Figure 3. identifies the work of Greeno (1978) and Greeno and Simon (1986). General task types are identified along with the skills and knowledge needed to address the task successfully. For example, problems of 'inducing structure' require a "process of apprehending the relations present among the problem elements and a process of generating an integrated representation of the pattern" (p. 244). This approach maps types of task demands directly onto skills which are required by the problem-solver in order to address the task successfully. This classification scheme is typical of cognitive science research in that it attempts to identify aspects of the environment with cognitive structures and processes used by the problem-solver. This type of rigorous mapping is missing in the AI findings above. There are restrictions, however, on the use of this research based on the methodology used to develop the classification systems. The standard method is for the researcher to select a simple task with a very restricted and well-defined set of task (objective) cues (eg.Tversky and Kahneman; 1974; 1981) which the subject uses to solve the task. Depending on the 'success' or 'failure' to solve the task, certain cognitive processes are postulated to account for the results. One problem is that the researcher assumes he/she knows the salience of the task cues, or rather, assumes that the subject is using cues from the defined set to solve the problem. This aspect has been criticised by Margolis (1987). Also, the tasks upon which the results are based are highly restricted and laboratory-confined. The direct generalization of results to the area of knowledge acquisition requires caution. Although these criticisms suggest caution, the outcome of this research takes the form of what is needed in knowledge acquisition.

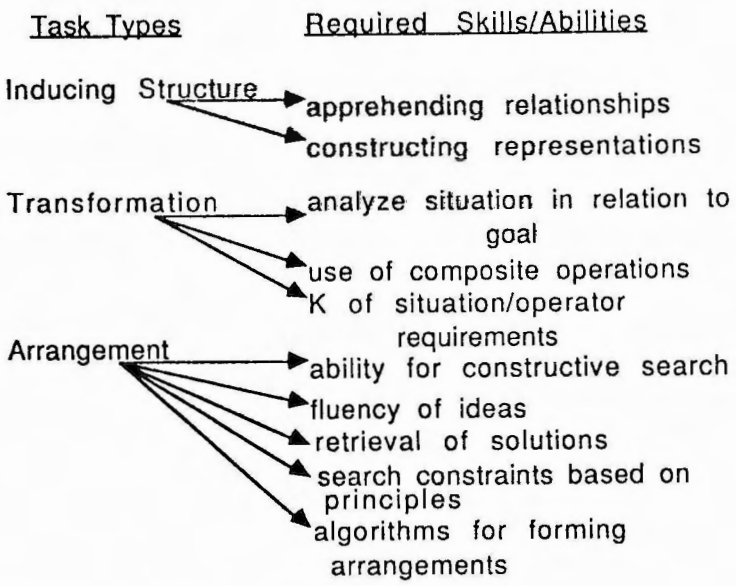


Figure 3. Greeno Taxonomy/Typology

3. Philosophy

There exist numerous examples of epistemological systems in philosophy which provide a justification and rationale for classification. Churchman's (1975) book on inquiry identifies five major schools of western philosophy which provide the basis for different types of problem-solving systems. Each system is characterized by its definition of knowledge, how the knowledge is combined, what constitutes a solution and the differing strengths and weaknesses of the inquiry process. Lockian, Leibnitzian, Kantian, Hegelian and Singerian inquiring systems are outlined.

The problem with these types of classification schemes is that they appear to be too conceptual in nature with little practical application, especially for our knowledge acquisition purposes. Churchman's scheme has been translated into a more practical tool which attempts to identify an individual problem-solver's mode or style of inquiry (Harrison and Bramson,1982). This tool has not yet been used in knowledge acquisition to help identify aspects of the problem-solver which influences knowledge elicitation and knowledge modelling.

Philosophical frameworks which have been developed into a knowledge acquisition framework are those of Brachman (1979) and Sloman (1980). The former identifies levels of representational primitives for different types of knowledge and the latter suggests four levels of depth at which questions about knowledge can be asked.These two frameworks have been used in the development of the KADS methodology (Hayward et al, 1986). A variety of interpretation models have been built using this methodology. Figure 4. is a replication from Breuker et al (1987, pg. 48)

This partial taxonomy is based on a particularly complex framework and methodology. The list in the figure is that of 'problem-types' so it is difficult to determine if these types are indicators of the task or the problem-solving method, or both. From the developmental history of the classification framework, it would appear that both the problem-solver and the task demands are accommodated through the different levels of knowledge inquiry. The mapping of task demands to problem-solving strategies is apparent through its top-down approach to development but the mapping methodology over the four levels of knowledge is not always explicit

The 'Task-Method' as an Integrating Concept

The representative taxonomies presented above do not fully offer a solution to the task-problem solving method dichotomy. Developing a taxonomy or classification index for task types alone or for inferencing or problem-solving methods alone offers little advantage and may even appear misleading for development of the field of knowledge acquisition. The

- system analysis
  - identify
    - classify
      - simple-classification
      - diagnosis
        - single-fault-diagnosis
          - heuristic classification
          - systematic diagnosis
            - causal-tracing
            - localization
        - multiple-fault-diagnosis
    - assessment
  - monitor
  - predict
    - prediction-of-behaviour
    - prediction-of-values
- system modification
  - repair
  - remedy
  - control
    - maintain
- system synthesis
  - transformation
  - design
    - transformational design
    - refinement design
      - single-stream-refinement-design
      - multiple-stream-refinement-design
    - configuration
  - planning
  - modelling

Figure 4 Taxonomy of Problem-Types (Breuker and Wielinga, 1987)

approach leaves out the problem solver or, more abstractly, the task-solving method used by the problem solver to address and satisfy the task demands successfully. Rather than classify the 'objective' task demands in quasi-objective terms, a more fruitful, an perhaps

more accurate, approach is to define the 'task' in terms of the task-solving methods used to address the task successfully. The definition of the problem-solving situation becomes operational and utilitarian because it has been established by recognizing and defining those methods problem solvers use to satisfy the perceived task demands. The 'task' and 'problem-solving method' become a 'task method' which is defined as an interaction between the task demands and the method(s) used to address them. The 'task' is modelled together with the methods and procedures used to solve it. Task-methods, then, become the raw material for an eventual taxonomy of task-situations.

The goal of this paper is to develop criteria for 'task-methods' so they will be useful and meaningful for knowledge acquisition. In order to accomplish this goal, a framework is required which recognizes the important aspects of knowledge acquisition and within which the task-method can be operationally defined.

A Conceptual Framework for a Task-Method Development

In order to define a 'task-method' perspective, the recognized components need elaboration and discussion. The task-method system is composed of the 'task', with its structures and patterns, the procedure(s) and models(s) to address the task successfully, and the interactions of the two. Recognizing and elaborating the task-method to building a final operational model (a knowledge based system) requires comprehension of knowledge acquisition activity. Norman (1983) points out that the first requirement is a target system existing in the real world which consists of an organized structure (see Figure 5). This system acts as the focal point for the human problem solver (sometimes an expert problem solver). No evident model exists in the target system, but rather an ordered or repetitive sequence of activities or events. A variety of potential models are possible from this target system (t). This target system represents the 'task'.

The first modeller interacts with (t) and may produces many useful models and may display many modelling processes. This perceiving system is capable of isolating and defining "primitives" by representing them in symbolic form. There is some argument in cognitive science about the nature of these representations and about the level at which the representations are confirmed. This research suggests that it is quite possible for various models to exist at various levels in the perceptual and cognitive levels (Pylyshyn, 1984; Fodor, 1980;and Paivio, 1977) . Norman (1983) has called models generated by the human as M(t)s or models of the target system (t).

The next model and modelling process results in an M(M(t)) or a model of the model of the target. Norman refers to this as the conceptual model of the target because it is the model that is reflected in communicating the M(t) to other perceiving entities. Norman represents this model as the conceptual model C(t). The C(t) finds representation in the communication primitives of language and behaviour. The M(t) affects the form and representation of the C(t) and is, in turn, constrained by the language and behavioural representational system. It is this C(t) that is available to others and the one which requires further translation and representation by other modelling activities. As the M(t) is unavailable to an observer, the C(t) is taken to represent the model used to address the task demands (t).

The C(T) becomes the focus of further modelling efforts. Knowledge engineers take these rather sloppy extremely rich and expressive models and enforce an order. The C(t)

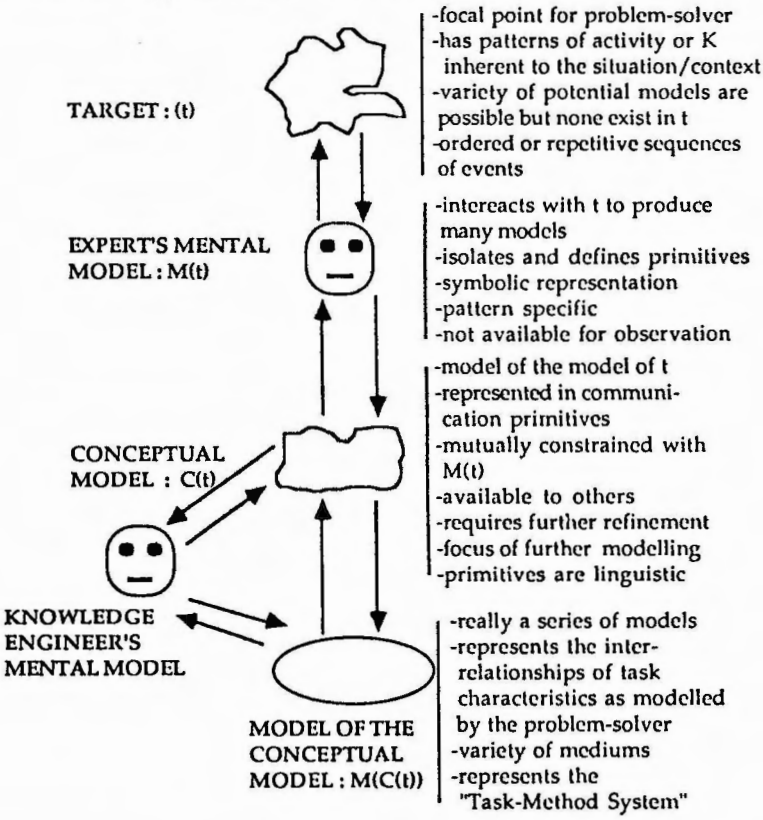


Figure 5. Knowledge Acquisition Process

gives up its set of primitives as recognized by the analytic methods used (which, of course, are underlined by a model of their own). The C(t) becomes the the target (t) for the knowledge engineer. The whole point of this modelling activity is to select a set of primitives from the C(t) which can be used for another model. But the activity requires an ordered and systematic procedure of its own.

Representing the primitives of C(t) necessitates another medium and another representation structure. What occurs here is actually a series of models but essentially they are all M(C(t))s, or models of the conceptual model of the target, in various forms of refinement and representation. The medium may be paper using a variety of coding schemes or the models may exist in machine form such as rules, frames, semantic nets, etc. These models, too, are usually designed to represent different primitive types, which are considered important components. Primitives, which represent facts of the domain, are included with the inferencing procedures. Also, forms of strategic processes which guide the use of domain knowledge primitives and the inferencing processes are included. The result of this activity is a new model of the C(t) which is the M(C(t)). This model represents the interrelationships between the task demands (t) and the the model for addressing the demands C(t). It represents the salient features of the task and its context.

Each successive model requires a representational schema, another medium of representation, and an ordered procedure for translating one representation to another. The M(C(t)) is thought of as containing the necessary primitives to recreate the operation of the M(t). Also, there may be many M(C(t))s that can perform the same activity of simulating the M(t). The M(C(t)) represents the final working or operational model: it may be in machine codeable form or on paper.

Criteria to Exploit M(C(t))s as an Operational Definition of Task-Methods

M(C(t))s present the basis for the final knowledge base or they may actually represent the final system. All share certain qualities such as the necessary knowledge primitives represented within a formalism and all have resulted from a process of systematic effort. M(C(t))s are assumed to be as representative as possible of a conceptual model C(t) or a group of C(t)s.

1. The M(C(t)) determines and identifies for selection the knowledge primitives from the C(t). Not only do M(C(t))s accurately display these primitives, but they can be used to look for, locate and isolate, and define those primitives. They can act as pointers to determine the focal points in the C(t). If properly and effectively described, these models can help recognize the operational knowledge in the C(t). As well, the M(C(t)) identifies elicitation methods which are as free from bias as possible and which are accurate methods for eliciting and representing primitives. The elicitation procedure(s) suggested by the M(C(t)) are accurate and faithful methods for gathering the required primitives for the M(C(t)). Bias enters when the procedure itself begins to impose a structure on the process which reduces the possibility of obtaining what is required.

2. The M(C(t)) maintains the task/method connections represented in the C(t).

Salient cues in the task environment are linked to knowledge structures used by the problem-solver. The problem-solvers methods of processing knowledge from the environment and from memory are identified and maintained. The links or connections are represented in a manner which is verifiable and open to validation.

3. The M(C(t))s are easily and quickly developed. Because M(C(t))s have the potential to guide all phases of knowledge base development, the sooner they are defined and available, the sooner they can help organize and structure subsequent procedures such as knowledge elicitation and acquisition activity. Initial elicitation required to identify the M(C(t)) should be relatively unobtrusive and easily accomplished. At these initial stages, the C(t) with very little manipulation acts as the basis for M(C(t)) identification.

4. The methods for developing the M(C(t)) are empirically based. If these models are defined as objectively as possible, then there can be little misunderstanding of their definition and the models can be further studied and developed. The methodology used in developing the M(C(t))s needs to approach algorithmic form. This would ensure that the procedures used to develop these models are replicatable.

5. A shared representational language or modelling framework for M(C(t))s allows empirical comparisons across models. Each M(C(t)) is produced using a common representation system so the resulting models can be categorized, compared and differentiated. This possibility makes it much easier to recognize different M(C(t))s for the same C(t) and provides a common language for discussion. Characteristics of the models act as a basis for comparison and classification.

Comparison of Knowledge Acquisition Methodologies

This section of the paper addresses three current approaches for developing M(C(t))s (Rapid Prototyping, Generic Structures, and Knowledge Analysis). Each shares the essential qualities mentioned above; however, each has its own unique characteristics which offer direction and guidance for their use. Figure 6 offers a summary of the following discussion.

Rapid Prototyping

The concept of rapid prototyping is based on the assumption that the building of knowledge based systems is inherently experimental (Hayes-Roth et al, 1983). Through a trial-and-error method, a knowledge base is eventually constructed to meet requirements. Usually, the final representational formalism is selected prior to the elicitation procedures. These formalisms act as the the M(C(t)). Another method which may be considered a rapid prototyping approach to M(C(t)) development is induction. Induction type M(C(t))s require numerous examples of C(t) both usual or routine and unusual or peculiar. An algorithm (eg. ID3, Quinlan, 1979) is used to identify patterns in variable interaction in the case examples and represent them in a structures such as a decision tree. The tree can be further converted into a rule representation format (Hart, 1987). The resulting M(C(t)) can be further refined and developed to give the desired performance.

Criteria 1 and 3

Using a 'shell' as an M(C(t)) has its advantages and disadvantages. The shell (or AI language) specifies very clearly what primitives are required for the M(C(t)) but they do not provide clear pointers to the required primitives in the C(t) which, in turn, do not help define the elicitation procedures required at various stages of development. In many ways, the 'shell' is its own knowledge acquisition tool but the accurate display of the primitives in the M(C(t)) is questionable. However, knowledge bases can be developed rapidly and improved through successive iterations. The M(C(t))s may be discarded if inadequate in favour of better ones as more understanding develops from the experimental process.The induction method solves some of these disadvantages.Similar to the shell method, the induction method uses a pre-set formalism for the M(C(t)) (shell or decision tree and rules). This formalism acts to select the primitives from the C(t). The use of actual case examples provides an immediate resource of the C(t) from which patterns can be mined and although there are many caveats on these systems (Hart, 1987), they are generated quickly.

Criterion 2

The direct jump from C(t) almost directly to a machine form of M(C(t)) severely reduces the scope of the M(C(t)).Both methods separate the connections in the C(t) between the task characteristics (domain knowledge in rules) and the problem-solving

	Criterion 1 Identifies K Primitives	Criterion 2 Task/Method Connections	Criterion 3 Ease of Development	Criterion 4 Empirical Development	Criterion 5 Shared Representation
Rapid Prototyping Induction	-primitives for M(C(t)) -low identification in the C(t) -require "rule-sighting" -accurate display of primitives questioned -does not identify elicitation methods *induction solves above problems to some extent	-direct jump from C(t) to code reduces scope of the M(C(t)) -separates connections in C(t) -rules and inference -limited by K-represent. format and inferencing methods -good for very well defined problem domains	-developed very quickly -multiple iterations of increased refinement -shell is its own elicitation tool -working model with minimal knowledge acquisition	-experimentally developed -methodology often ideosyncratic -no set of formalisms for rule-building or for control knowledge -requires hacking	-share common k-rep. (rules/objects/frames) -share common inference -meaningless comparisons -impossible to generate useful taxonomy
Generic Structures (imitation)	-identifies and defines the primitives in C(t) -KA tool developed exclusively for problem -two levels of K addres'd	-links task variables with imitated process patterns -intermediate step before KA and KB devel	-M(C(t))s easily and quickly developed if the task -method is in the library	-formal method for control K development -development of new PSMs not well documented	-recurring procedures -useful when the task demands are known -but too specialized -partial taxonomy
Generic Structures (emulation)	-identifies processing -specific KA for each task type -two levels of K are addressed	-links task variables with PS process patterns -captures cognitive structure and process	-takes considerable time if no match with library	-resolution of the interaction problem demonstrates formalisms for development	-distiguishable modules -all independent of each other -allows comparison -typology exists
Knowledge Analysis	-finds many levels of primitives in C(t) -elicitation methods very restricted -method does not take advantage of guiding elicitation -elicitation follows same path for all tasks	-maintains very rich connections -four levels of K -objects and k-sources -paper representation less constraining than machine representation -some difficulty maintaining connections when coded	-easy to develop M(C(t))s if problem situation matches one in library -developing a new M(C(t)) would take time and method not well documented	-epistemologically developed then experientially tested -KEng judgment requ'd in developing Interpretation models -method becoming more formalized	-interpretation models all of the same rep. -easy to compare on the basis of K-sources and strategies -partial, but detailed taxonomy exists

Figure 6. Comparison of Three Knowledge Acquisition Approaches

methods (inference). This approach is severely limited by the knowledge representation formalisms and the general-pattern inferencing processes. Consequently, limited, well specified problems may be solved with this approach. The C(t) needs to be small and well-defined in order to fit the pre-defined M(C(t)).

Criteria 4

This approach is considered experimentally based (i.e. bottom-up), with new knowledge added and sometimes new directions taken on the basis of the previous attempts at constructing M(C(t))s. The methodology of developing formalisms (rules, frames, etc.) directly from linguistic material (with little analysis) is often idiosyncratic (Johnson, 1987) with few formalisms identified for rule-building at the domain level and especially for meta-level or control knowledge.

Criterion 5

M(C(t))s developed using this approach result in a shared representational framework. The resulting models are shell-based and in the form of rules, frames, etc. or are represented in an AI language (lisp, prolog). In this sense, they share a common representational formalism but only for one type of knowledge of the C(t). Domain level and inferencing processes are easily represented here but the M(C(t)) also requires control or strategic knowledge. Shell-based models provide this knowledge but in an ad hoc manner and induction systems have difficulty generating it. Also, it would be difficult to compare rapid prototyping M(C(t))s because the level of comparison is the detailed representation of the rule or frame. Comparisons here would be superficial at best (eg number of rules, etc.). Comparing the M(C(t))s on the basis of the general purpose inference engines in the shells would be equally meaningless. Attempts at classifying general inference strategies and ad hoc control structures would result in a taxonomy of marginal use.

Generic Structures

Because of some of the limitations with the rapid prototyping method, researchers have developed methodologies which attempt to understand the demands of particular task situations, to recognize the similarities and differences of these situations, and to construct functional models which help to guide knowledge acquisition and ultimately, the resulting knowledge base. Two different approaches to building the M(C(t)) have emerged. The first may be called an 'imitation' method (McDermott, 1988a) and the second the 'emulation' method (Chandrasekaran and Bylander, 1988). The underlying assumption in this approach is the existence of recurring patterns of knowledge use in a variety of problem solving methods for a variety of task demands. These patterns can be made explicit and then combined to meet different task requirements.

The goal of the imitation method is to "select problem solving methods and tools to fit the demands of the task" (McDermott, 1988a, p. 226). This is accomplished by 'exploiting' the structure of the 'task' in order to automate knowledge acquisition. The underlying assumption is that this methodology and the resulting tool act as a theory of how knowledge may be modelled or recreated.

The prime concept is that of the 'problem solving method' (PSM) which is characterized as an objectifiable and describable procedure of "identification, selection, and implementation of a sequence of action which meets the requirements of a task solution" (McDermott, 1988a, p. 225). Once specified, the PSM displays important attributes and uses. It provides a means of identifying possible or "candidate" actions at each step in the task solution sequence. Expectations are developed against which the selected actions are compared. McDermott points out that knowledge roles can be objectively characterized in an algorithmic form which specifies when to use what knowledge and what knowledge is needed for making the proper selections of the knowledge roles. PSMs are assumed in all tasks but that they may not be easily specified because the control knowledge essential for identifying the recurring roles is too 'bound' in the domain knowledge. Those which can be identified are called 'role limiting methods' which are general or weak PSMs that define the task specific control knowledge. The role limiting method defines the roles of the task specific or domain knowledge and gives specific recommendations for the knowledge representation formalism. Examples of a role limiting method are the cover-and-differentiate method of MOLE (Eshelman et al, 1987) and the propose-and-revise method of SALT (Marcus, 1987). Each consists of a recurring sequence of 5 to 10 actions on domain knowledge which imitates the way a human solves the problem.

The other methodology for developing a task-level language at an organizational level above the symbolic level of the rapid prototyping approach is that of Chandrasekaran (1988). This method assumes that there are "significant aspects of cognition and perception [that] are usefully understood as information processing activities on representations" (Chandrasekaran, 1987; p. 4-0) and that these significant aspects can be objectified and modelled or 'emulated'. Generic Architectures (significant information processing patterns) are identified for a particular set of task demands. For example, a certain type of diagnostic task may be made up of the process of classification, then abstraction, then classification of the abstractions. Another diagnostic task may require an additional generic architecture. The architectures act like the role limiting methods in that they help to identify the various knowledge processing "blocks" and can order knowledge acquisition.

Criterion 1

The two methods of the generic structure approach provide improvements on the rapid prototyping approach. If a generic architecture or role limiting method can be distinguished early enough in knowledge acquisition, it can guide subsequent knowledge elicitation and knowledge representation activity. This approach identifies and defines the primitives in the C(t) and the patterns of processing peculiar to the task demands (two levels of knowledge). Knowledge primitives are carefully identified and elicited at the proper level of abstraction (Bylander, 1987). It is unclear in both methods, however, the decisions which determine whether a generic structure applies or not or how one is distinguishable or not. It can be assumed that patterns are recognized by the knowledge engineer or researcher and then this pattern is compared to known, previous patterns. Knowledge acquisition procedures are often developed specifically for the task situation identified. For example, Musen (1988) built PROTEGE which develops knowledge acquisition tools for the task of skeletal-plan refinement.

Criterion 2

The methodologies of this approach attempt to maintain the richness of the C(t). Task variables are linked either to objectively established process patterns (imitation) or to problem-solving process patterns (emulative).Both methods maintain the task/method connections and offer an intermediate representation between linguistic material and a coded knowledge base.

Criterion 3

In terms of speed of M(C(t)) development, both methodologies offer a "library" of M(C(t))s for initial comparisons. However, when recurring patterns in a particular task do not match any in the library, a good deal of time is required for identification and development of the role limiting method or generic architecture.

Criterion 4

The generic architectures and the role limiting methods represent rigorous instantiations of M(C(t))s at the knowledge level (Newell,1982). Both methods in this approach are empirically based and have developed formal methods for identifying control knowledge. As well, the development of new M(C(t))s is well formulated and documented.

Criterion 5

With respect to the use of shared representational formalisms to promote empirical comparisons across tasks, the role limiting methods appear too specifically and idiosyncratically constructed to provide a useful comparison. Individual steps in the propose-and-revise method do not appear to be re-useable in other task situations. The generic architectures, however, identify clearly distinguishable 'modules' of activity which remain intact regardless of their use or combination. McDermott (1988b) has developed a partial taxonomy of existing role limiting methods. The generic architecture approach appears, at this point, to be more of a partial topology of M(C(t))s.

Knowledge Analysis

This approach to the development of M(C(t))s is characterized as a multi-level analysis of the task or problem space and the problem solver. The approach assumes that knowledge is multi-level phenomenon and that knowledge acquisition must take this into account. The primary concept is similar to the task specific approach but extends beyond the single level analysis of this approach. At the heart of this approach is the Interpretation Model (Breuker & Wielinga, 1987). Interpretation models consist of five components. The first is the identification of the input and output object classes required by the problem solver (eg data, hypotheses, actions, plans).Based on these, the type of task can be partially determined. For example, diagnostic tasks require symptoms, tests, fault classes, complications as objects (Breuker & Wielinga, 1987, p. 25).

The second component is that of knowledge sources. These configurations reflect a set of knowledge which infers or produces new or other objects. They are inference associations which use given objects to identify and order other objects. This approach suggests that these structures can also help determine the task type. The third component of the interpretation model is those 'model' or personal theories which the expert uses in

problem solving. These models may be causal, process, formal or empirical, or spatial in nature. They may be rigorous, objectively specified models or subjective, qualitative models.

Those structures which order the objects and knowledge sources in some framework (eg. types of hierarchies, sequences, relational networks) represent the fourth component. They are used to represent patterns of objects as they interact with knowledge sources. The final component is that of a strategy which represent the plans for using knowledge sources and structures. Similar to the use of control knowledge in the task specific approach, this level of analysis identifies the uses and roles of different knowledge types. In fact, well fixed or clearly identified strategies are called problem solving methods (Breuker & Wielinga, 1987, p. 35). To solve a task, an assembly of interpretation models are required.

Criterion 1

Knowledge primitives from the C(t) are identified systematically at many levels. This approach points to the knowledge primitives in the C(t) which are required by the multi-level analysis. However, despite the richness of the modelling method, little attention is paid to the development and selection of elicitation methods to match the levels of knowledge apart from various interview methods and protocol analysis.This approach assumes that all the necessary knowledge forms can be gathered by these methods, despite the reported drawbacks of these knowledge elicitation approaches (Wellbank, 1987) . The advantage of structuring or guiding the knowledge elicitation is not well exploited by this approach.

A main point of difference between the task specific approach and the knowledge analysis approach appears in the extent to which the two consider knowledge elicitation important. The former approach clearly identifies the important of building knowledge elicitation tools specific to the task type. The emphasis is to limit the role of the expert to the provision of domain or task-specific knowledge. The latter approach tends to involve the expert to a much greater degree so the resulting knowledge base reflects the benefits of the multi-level analysis of the knowledge. However, this approach relies on one knowledge elicitation method with an emphasis rather on the analysis methods.

Criterion 2

The interpretation model is used as a guide to map the C(t) into symbol level formalisms by way of a multi-level M(C(t)). This approach is able to model more of the various knowledge primitives and therefore, more of the richness, of the C(t) than the other two approaches. The four levels of knowledge provide ample opportunity to maintain the task/method connections. This appears to be done through the objects (task characteristics) and knowledge sources (problem solving methods). It is not clear whether the connection scan be maintained once the paper M(C(t)) is translated into code.

Criterion 3

The interpretation model does not appear to be quickly or easily developed. The extensive multi-level analysis is time consuming. If the structure of the task can be matched to existing interpretation models, then this time is reduced. This approach minimally structures the knowledge elicitation and concentrates on the analysis of the elicited protocols and interviews.

Criterion 4

This approach develops M(C(t))s epistemologically and then tests them experientially. The procedures are complex but appear generally formalized and replicatable. Knowledge engineer judgment is required at many stages of the development of M(C(t))s which suggests that more formalism is required.

Criterion 5

A shared representational language appears evident for this approach. Interpretation models, like task specific architectures, can be compared and re-arranged to fit the situation. The common representational scheme allows comparison at the level of the knowledge sources and strategies. A well-developed, but as yet incomplete, taxonomy exists.

Future Work

The next step in the research is to develop operational refinements of each of the criteria so that they may act a outcome evaluation criteria. These criteria would aid in the development of a methodology which develops M(C(t))s (task-methods). The methodology requires fomalized procedures, sequential representational formats and an opportunity for automation of the steps in the methodology. The three approaches outlined above offer valuable procedures and representational tools towards the development of a task-method methodology.

References

Anderson, J.R. (1983) **The Architecture of Cognition.** Cambridge, Mass: Harvard University Press.

Boose, J.H. (1988) A survey of knowledge acquisition techniques and tools. In the **Third Knowledge Acquisition for Knowledge Based Systems Workshop, AAAI:** Banff, Canada.

Brueker, J. and Wielinga, B. Use of models in the interpretation of verbal data. In **Knowledge Acquisition for Expert Systems: A practical handbook.** (ed) Kidd, A.L.New York: Plenum.

Brachman, R.J. (1978) A structural paradigm for representing knowledge. **BBN Technical Report,** Cambridge, MA: Bolt Beranek and Newman Inc.

Bylander, T. and Chandrasekaran, B. (1987). Generic tasks for knowledge-based reasoning: the "right" level of abstraction for knowledge acquisition. **International Journal of Man-Machine Studies,**Vol. 26 (2), 231-244.

Chandrasekaran, B (1988) Generic tasks as building blocks for knowledge-based systems: the diagnosis and routine design examples. **The Knowledge Engineering Review,** Vol. 3,3, 183-211.

Chandrasekaran, B. (1987)What kind of information processing is intelligence? A perspective on AI Paradigms and a proposal. In the **Second Knowledge Acquisition for Knowledge-Based Systems Workshop.** AAAI: Banff, Canada.

Churchman, C.W. (1971) **Design of Inquiring Systems.** New York: Basic Books.

Clancey, W. (1985) Heuristic classification. **Artificial Intelligence** 27(3): 289-350.

Eshelman, L., Ehret, D., McDermott, J. and Tan, M. (1987) MOLE: a tenacious knowledge acquisition tool. **International Journal of Man-Machine Studies**,Vol. 26 (1), 41-54.

Fodor, J.A. (1980) **Representations.** Cambridge Mass.: MIT press.

Greeno,J.G. (1987) Natures **of problem solving abilities. In Handbook of Learning and Cognitive Processes, Vol. 5.(ed)** Estes, W.K.

Greeno, J.G. and Simon, H.A. (1986) Problem solving and reasoning. In **Steven's Handbook of Experimental Psychology,** (eds) Atkinson, R.C., Herrnstein, R.J., Lindsey, G. and Luce, R.D., New York: Wiley.

Harrison, A. and Bramson, R.M. **Styles of Thinking.** New York: Doubleday.

Hart, A. (1987). Role of induction in knowledge elicitation. In **Knowledge Acquisition for Expert Systems: A practical handbook.** (ed) Kidd, A.L.New York: Plenum.

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

Hayward, S.A., Wielinga, B.J., and Breuker, J.A.(1987) Structured analysis of knowledge. In the **Second Knowledge Acquisition for Knowledge-Based Systems Workshop.** AAAI: Banff, Canada.

Johnson, N.E. (1987) Knowledge elicitation in practice. **Insight Study No.6.** Systems Designers Scientific, Pembroke Broadway, Camberly, Surrey, UK.

Kidd, A.L. (1987) **Knowledge Acquisition for Expert Systems: A practical handbook.** New York: Plenum Press.

Kitto, C.M. and Boose, J. Selecting knowledge acquisition tools and strategies based on application characteristics. In the **Second Knowledge Acquisition for Knowledge-Based Systems Workshop.** AAAI: Banff, Canada.

Marcus, S. (1987) Taking backtracking with a grain of SALT. **International Journal of Man-Machine Studies,** Vol. 26 (4),383-398.

Margolis, H. (1987) **Patterns, Thinking, and Cognition: A Theory of Judgment.** Chicago: University of Chicago Press.

McDermott, J. (1988a) Preliminary steps toward a taxonomy of problem solving methods. In **Automating Knowledge Acquisition for Expert Systems,** (ed) Marcus, S. Kluwer Academic Publishers: Boston.

McDermott, J. (1988b) Exploiting task structure to automate knowledge acquisition. **Invited Address, AAAI,** St. Paul.

Minsky, M. (1975) A framework for representing knowledge. In P.H. Winston (Ed.), **The Psychology of Computer Vision.** New York: McGraw Hill.

Musen, M.A. (1988) An editor for the conceptual models of interactive knowledge acquisition tools. In the **Third Knowledge Acquisition for Knowledge Based Systems Workshop,** AAAI: Banff, Canada.

Newell, A. (1982) The knowledge level. **Artificial Intelligence,** 18, 87-127.

Norman, D.A. (1983) Some observations on mental models. In Gentner, D. and Stevens, A. (eds.), **Mental Models.** Hillsdale, N.J.: Erlbaum.

Nystedt, L. (1983) The situation: a constructivist approach. In **Applications of Personal Construct Theory.** New York: Academic Press.

Paivio, A.U. (1977) Images, propositions and knowledge, in Nicholas, J.M. (ed.) **Images, Perception and Knowledge:** Reidel.

Pylyshyn, Z.W. (1984) **Computation and Cognition.** Cambridge, Mass.: MIT Press.

Quinlan, R. (1979) Discovering rules from large collections of examples: A case study. In **Expert Systems in the Microelectronic Age,** (ed) Michie, D.Edinburgh University Press: Edinburgh.

Schank, R.C. & Abelson, R. (1977) **Scripts, Plans, Goals, and Understanding.** Hillsdale, NJ: Erlbaum.

Schank, R.C. (1982) Reminding and memory organization: an introduction to MOPs. In W.G. Lehnert & M.H. Ringle (Eds), **Strategies of Natural Language Comprehension.** Hillsdale, NJ: Erlbaum.

Sloman, A. (1980) **The Computer Revolution in Philosophy,** Harvester.

Sowa, J.F. (1984) **Conceptual Structures: Information Processing in Mind and Machine.** Menlo Park, California: Addison-Wesley.

Tversky, A., and Kahneman, D. (1974) Judgment under uncertainty. In Kahneman et al **Judgment Under Uncertainty: Heuristics and Biases.** Cambridge: Cambridge University Press.

Tversky, A., and Kahneman, D. (1981) The framing of decisions in the psychology of choice. **Science,** 211:453-58.

Welbank, M. (1987) Knowledge acquisition update: **Insight study #5.** Systems Designers, Pembroke Broadway, Camberly, Surrey, UK.

Winston, P.H. (1984) **Artificial Intelligence (2nd ed).** Reading. Prentice-Hall.

Situation Pattern Acquisition
Improves the Control of Complex Dynamic Systems

Guy A. Boy [1,2,3] *& Thierry Caminel* [2]

[1] Groupe d'Intelligence Artificielle
Centre d'Etudes et de Recherches de Toulouse
Office National d'Etudes et de Recherches Aerospatiales
2, avenue Edouard Belin
31055 Toulouse Cedex
France

[2] Dialogics S.A.
Le Prologue, Voie 1
BP 213
31328 Labège Innopole
France

Third European Knowledge Acquisition ECAI Workshop, Paris, France, July 3-7, 1989

Keywords. Knowledge-based operator assistant systems. Knowledge acquisition and representation. Space telemanipulation application.

Abstract

This paper describes an approach to the problem of capturing Operator's knowledge -- the problem of defining an adapted knowledge representation (KR) useful for capturing, storing, and using knowledge about an engineered system to be controlled. Problems are represented by a problem statement (situation pattern) and a solution which is a trace of a problem solving process. The more formalized and complete the problem statement, the less the problem solution will involve complex problem solving techniques, and conversely. Most of the work already done in AI has been focused on problem solving techniques. Results of this work have been applied in expert system (ES) technology. Consequently, good ES are not at present representing good human experts but rather good problem solvers in a narrow application domain. Situation pattern acquisition (SPA) is incremental. This technique consists of two phases: elicitation of the a priori system analysis and designer knowledge, and the incremental process of SPA by experimentation. Some of the knowledge structures produced in a space telerobotics application are given in this paper. In order to generalize this method, criteria for a successfully deployed application are also given. This technique has been very successful so far for our space application.

1. Definition of the problem: Operator Assistant Systems

Automation has been one of the main issues in engineering for several decades. Engineers and scientists have focused on design and optimization of automated systems, e.g., autopilots of aircraft, satelitte launching, nuclear power plant control, etc. Most of this work has not taken into account the human in the loop. So far, while there have been few incidents and accidents on such systems, most of these have been shown to be human-dependant (Nagel, 1986). Thus, human operators become important limiting factors in the interaction loop with complex and reactive systems. The actions of operators are guided by mental models, i.e., executable schemas which structure their knowledge and lead to considerable economy in planning. While these mental models are very useful in well learnt and recognized situations, in novel situations

[3] Present address: NASA-Ames Research Center, Artificial Intelligence Research Branch, Mail Stop 244-17, Moffett Field, California 94035, USA.

they may fail due to the incompleteness of the available data. This means that human operators may fail to recognize the critical parameters and interpret new situations. Thus, too much reliance on mental models and highly learnt procedures can lead to human errors. Lack of knowledge comes from an incomplete understanding of system functioning. Of course, training can help to prevent such human errors. However, vigilance and workload problems necessitate the use of operator assistant (OA) systems.

Moreover, in abnormal situations with high workload, human operators can be unique problem solvers if they are well informed quickly of the actual world situation. Situation awareness is one of the major challenges for designing OA systems in problem solving tasks. Indeed, to provide the human operator with the "right" situation, it is necessary for the OA system to have very deep knowledge about how the system works and how it can be used. This deep knowledge may come from system analysis and from elicitation of expert designers' knowledge. It is also necessary to build a human operator model for designing the user knowledge to be introduced into the OA system and for evaluating it. At present, we are using a model called SRAR (Situation Recognition and Analytical Reasoning, see figure 1) which has been developed for representing human problem solving in dynamic environments (Boy, 1987). This model was derived from experimental results on a diagnosis assistant system developed at NASA (Boy, 1986). We are now refining the early concepts of situation patterns and analytical knowledge.

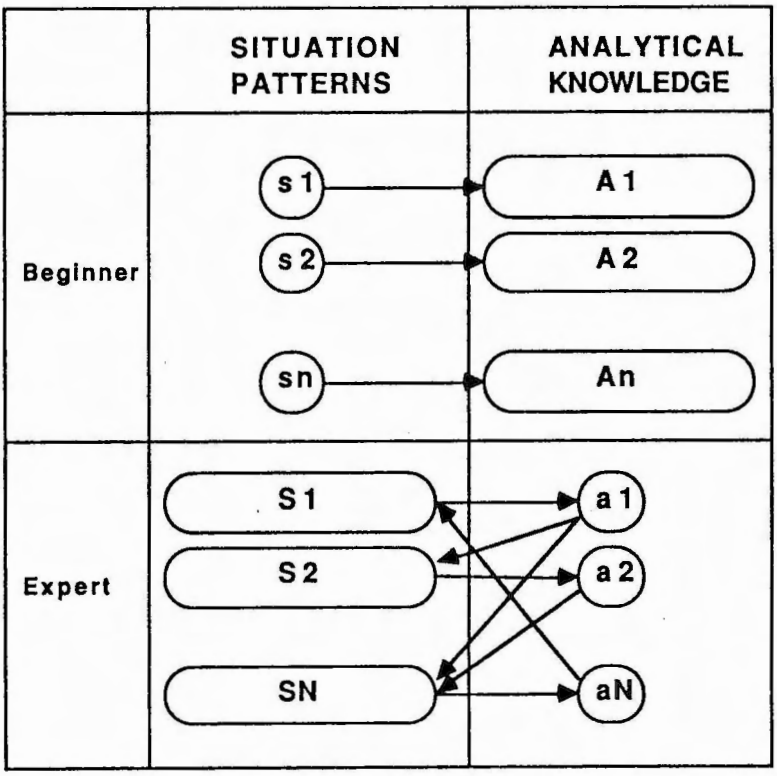


Figure 1: SRAR Model.

As a general rule, a problem can be represented by its "problem statement" and its "problem solution". Figure 1 illustrates the fact that the more formalized and complete the problem

statement (situation pattern), the less the problem solution will involve complex problem solving techniques (analytical reasoning), and conversely. However, human experts know how to state problems in order to match them with their own experience. They are good situation recognizers and generally solve domain-related problems much faster that anybody else. Taking Anderson's model, learning by training transfers analytical (cognitive) knowledge useful for beginner problem solving, into situational (associative) knowledge useful for real-time expert problem solving. In the associative phase, situational knowledge is generally represented by schemas or patterns.

To conclude, building OA systems leads to the following requirements: representing tractable complex patterns; compiling patterns and handling contexts in order to increase the situation recognition speed; changing reasoning strategy when necessary; and mastering knowledge transfer from an analytical to a situational form. This approach provides a tractable framework for nonmonotonic problem solving. We will now illustrate the above concepts on a real-world application: the remote control of a space robotic arm.

2. Description of the deployed application

The presented methodology has been developed for the last 8 years supported by seveal aeronautics and space applications. The OA system for space robotic arm remote control has been deployed since september 1986. It has involved five people for two years. A first mockup was developped in July 1987, using the Nexpert Object knowledge base building tool, Microsoft Windows and C language on a 386 PC compatible. It was refined and installed and connected to the robotic arm in January 1988. Users' knowledge acquisition and validation experiments were carried out throughout 1988. A prototype was installed late in 1988. We are currently working on the last run-time version of it. A research program was started in September 1987 on a blackboard architecture including an ATMS-like system (DeKleer, 1986) for nonmonotonic information processing investigations.

The system used is a 1/4 size model of a robotic arm to be used in the European space program. It consists of an adapted industrial arm, a simulation of the in-flight operating environment, and a satellite. The mission "grabbing a satellite in flight" has been chosen for the experimental tests for OA development. The human operator cannot see the robotic arm directly. He/she telemanipulates the arm through two joysticks and using two screens displaying information from three cameras in close and global vision. Graphic aids will also be available. Different functional modes are possible: manual, fully automatic, emergency, and mixed. The present approach is problem-driven (bottom-up), i.e., no theory is available a priori for building an appropriate OA system. However, a set of practical experiments have been carried out and the protocols analyzed. The corresponding results constitute a set of problems which are considered as a testbed for more theoretical investigations which will be presented in a later paper.

The prototype, called SAOTS, implemented for the CNES Robotics Laboratory, is an application of the general approach to OA systems described in the SRAR model. Situational knowledge is represented as patterns, or typical expected situations, while analytical knowledge includes objects describing the the robotic arm and the simulated environment, with rules allowing the control of the entire system.

The SAOTS prototype is running in real-time. It performs three main tasks. First, it analyses data coming from the robot and uses them for pattern matching and analytical reasoning. Second, it produces commands to the robot control computers which are simultaneously analyzing operator actions - commands can also be generated directly by the OA system, depending on the level of autonomy sharing with the operator. Third, it displays information on

the interface screen. It should be noted that the OA system interface was elaborated using a windowing system and graphics.

3. Innovation

Representation issues

Problem statements can be represented by situation patterns which include three components : a combination of conditions on the parameters of the world, a context in which this pattern is valid, and a deduced hypothesis for further problem solving. The conditions are tested if the current context matches the context attached to the situation pattern. At a given time, the set of active situation patterns present in the short term memory is called the focus of attention. At each instant, these patterns are matched to the percieved situation of the world. When a situation pattern is matched, a problem solving hypothesis is suggested. One pattern can be followed by both the machine and the human, while others are tested in the background. If disjunctive hypotheses are suggested, this approach permits on-line testing of whether the hypotheses are exclusive or not. This can be done by maintaining the consistency of all the facts deduced from the suggested disjunctive hypotheses. When a contradiction between facts is detected (by the operator or the system itself), the corresponding hypotheses are automatically recorded as exclusive by the system.

Problem solving knowledge is represented by a set of blocks which can be linked for further analytical reasoning. A block of analytical knowledge includes five fields: a goal to be reached, a context in which this block is enabled, a set of necessary conditions to be satisfied for starting any work with this block, a set of actions to be used for reaching the goal of the block, and a set of constraints to be satisfied for staying in the block until the goal is reached. A violated constraint leads to the activation of one or several situation patterns and/or contexts. Analytical blocks can represent either shallow or deep knowledge, depending on the level of detail and analysis.

Furthermore, introducing constraints at a lower grain of the knowledge representation (blocks) has been observed to be a natural way to make nonmonotonic links explicit in the knowledge base. Because the system must be incrementally updated with new patterns for selecting hypotheses, the only effective way to handle multiple and potentially contradictory hypotheses seems to be a global manager taking into account external world constraints (patterns) and semantic task constraints (specifying mutually exclusive hypotheses).

Incremental knowledge acquisition (KA) and learning issues

Humans tend to compile their initial analytical knowledge into situation patterns. This creates automatisms which are very efficient, e.g., which have almost instantaneous reaction times. This is due to the highly parallel pattern matching process. Using conventional computers, such a mechanism cannot be reproduced with the same level of performance. Indeed, if knowledge elements are tranferred from the analytic side to the situational side, the first effect is to reduce the search process (cognitive effect), but it also increases the pattern matching. This does not decrease considerably the overall computation time as in the human case. However, this learning mechanism allows designing of situation patterns organized by context. Thus, only a few patterns have to be scanned at a given time, simulating the focus of attention. Moreover, the situation patterns which are considered in the focus of attention can be compiled using techniques comparable with RETE-like methods (Forgy, 1982).

Two phases of KA have to be distinguished here: (1) elicitation of the a priori system analysis and designer knowledge, and (2) the incremental process of KA by experimentation. The former

deals with design knowledge, i.e., how the system being controlled works. The latter tries to formalize the user knowledge, i.e., how the system being controlled is effectively used. During the second phase, new knowledge is acquired, but the main problem is the tranformation of initial sub-optimal knowledge into a more sensible and tractable form. The SRAR model allows the formalization of a learning mechanism transferring elements from analytical blocks into situation patterns. The main difficulty is to get simultaneously an appropriate knowledge representation and implement experimental protocols for acquiring such knowledge.

Experiments have been carried out with astronauts manipulating the arm. Verbal protocols, videos, and the main parameters have been recorded and analyzed. This has been done under various operating conditions, e.g., normal and abnormal conditions. A first attempt using this knowledge acquisition method has been presented at the EKAW-88 (Boy & Nuss, 1988). A formal method has been proposed for the knowledge acquisition process (Boy & Delail, 1988), and has provided a body of refined user knowledge. It has led to a reformulation of the initial knowledge representation. In particular, analytical knowledge has been represented as blocks. Remote control phases are represented as contexts. It appears that these blocks of analytical knowledge were very shallow due to the natural integration and compilation of knowledge of the users and the designers. These experts have been shown to be very skilled and knowledgeable about both the system being controlled and the task being performed using this system. Thus, taking into account experimental knowledge only is very hazardous for satisfying safety requirements. Physical analysis of the system to be controlled must obviously complete the experimental knowledge acquisition.

Finally, incremental KA using the SRAR model has been shown to be interesting for designing situation patterns which could be directly displayed to and understood by the operator.

4. Knowledge Structures Produced

Results of our experiments showed that the knowledge of a telerobotics OA system should include the following structures (figures 2 and 3).

Mission. A mission is described by a goal (for instance, grab a satellite), a set of resources and a set of action principles to reach the goal.

Plan. A plan is a hierarchical process which orders a set of actions to reach the goal. For a given mission, several plans can be performed according to the available resources. A plan is a chain of plan-elements which will be called "blocks".

Operations. An operation is a practical realization of a plan.

Resource. A resource is a hardware entity which allows an operation to be performed. For instance, cameras allow visualization of part of the real world.

Action principles. An action principle is a software entity which allows an operation to be performed.

Configuration. A configuration is a set of available resources and actions principles which allow a mission to be carried out. If all the resources of the system are available, the configuration is said to be "normal". When a resource fails, the configuration is said to be "abnormal".

State. A state is a set of state variables, e.g., arm positions, target positions.

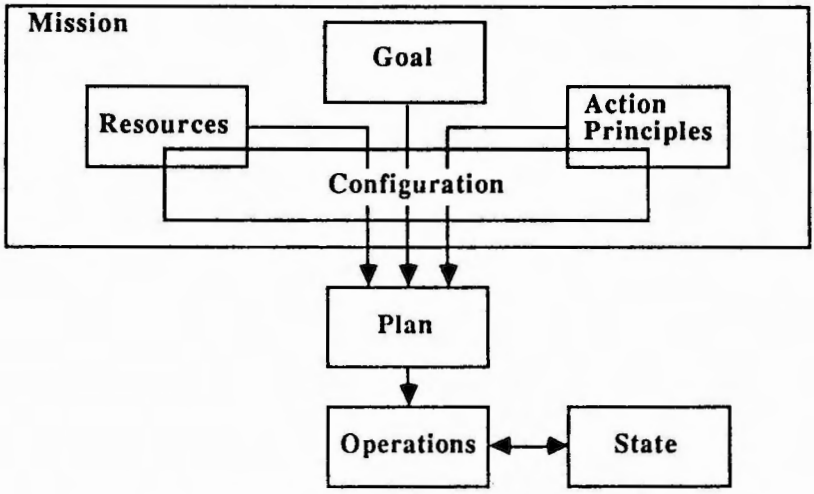


Figure 2: Knowledge organization for the telerobotics application.

Situation patterns. A situation pattern is a set of conditions on particular states and related by a logical operator. It corresponds to a problem statement. Situation patterns may be for instance: "lateral rotation close to joint limit", "elevation close to left joint limit and Y < 1 meter", "control law is PI", "$(Y-y)^2 + (Y-y)^2 < R^2$", or "$\partial x/\partial t < 50$ cm/sec".

Blocks. A block is a set of assertions which are always valid, stored and recalled at once. A block is a part of a plan. The three basic schemas for a block are: sequence, conditional branching, and loop. A block is defined by its initial conditions, final conditions or goal, constraints, and actions to be executed to reach the goal (Figure 3). A block corresponds to a given configuration, which, if it is no more valid, leads to an interruption of the block execution (constraints express this). Blocks may be for instance: "normal target detection", "target detection without the gripper camera".

Anomalies. An anomaly is represented by a violated constraint either leading to a new situation pattern (strong constraint) or conditional branching (weak constraint).

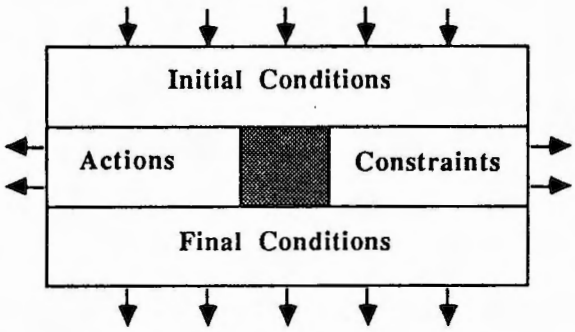


Figure 3: Graphical representation of a block.

Consistent set of blocks. A consistent set of blocks is a network of blocks which uses a minimal set of resources and action principles to reach a given goal. There is always a block which will start the deployment of a consistent set of blocks, initial conditions of this block will be called a situation pattern.

Strong constraints (of a block). A strong constraint is an anomaly which impose to move from one consistent set of blocks to another.

Weak constraints (of a block). A weak constraint is an anomaly which does not challenge the current consistent set of blocks. It corresponds to a conditional branching of blocks (Figure 4).

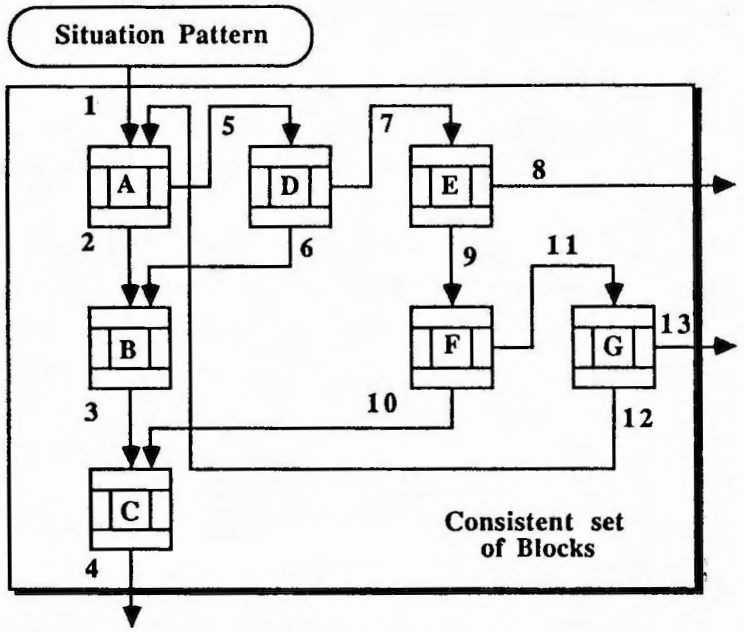


Figure 4: A situation pattern and a consistent set of blocks[1].

Phases. A phase is characterized by a consistent set of blocks leading to given final conditions (goal). A phase is basically hierarchical. A mission can be split into phases. For a given phase, there exists in general several consistent sets of blocks for reaching the goal. Phases can be for instance: "power up", "target detection", "grasping".

These knowledge structures are currently being developed towards a more theoretical framework for situational knowledge acquisition.

[1] In general, links between blocks can be: inputs into a consistent set of blocks, e.g., link 1; outputs from a consistent set of blocks, e.g., link 4; regular internal links, e.g., links 2, 3, 6, 9, 10; weak-constraint links, e.g., links 5, 7, 11; strong-constraint links, e.g., links 8, 13; loop links, e.g., link 12.

5. Criteria for a successfully deployed application

In a short term perspective, this approach leads to several technical issues. (1) Faster calculation speed is needed: real-time symbolic systems are still to be tested especially in reactive environments. (2) The more situation patterns are sophisticated, the less hypotheses have to be tested. This increases the efficiency of the corresponding problem solving. However, if many patterns have to be scanned at each time, an increased scanning time will lead to poor overall performance. Thus patterns have to be compiled. Methods of pattern compilation are currently being investigated. (3) Knowledge acquisition in reactive environments requires more systematic methods. In particular, machine learning techniques such as chunking (Laird, Newell, & Rosenbloom, 1987) have to be examined further.

In a long term perspective, the technique presented above needs to be adapted and implemented for multi-processor hardware. This is necessary for real-time performance and will improve asynchronous aspects of processing such as situation recognition, without affecting other synchronous processing. Furthermore, using multi-processor techniques allows the implementation of several inference engines, which could develop and test several hypotheses in a true real-time parallelism. Finally, computational primitives have to be further designed for a better implementation of the cognitive primitives (Rappaport, 1988) that we elicit from our experiments.

Finally, the most important criterion for successful applications is iterative experimental verification.

6. Payoff

This approach leads to a new methodology for designing operation guidelines and assistant systems for controlling complex dynamic systems. It is currently being tested on a real world application. The resulting OA system is being used in the robotics laboratory at CNES. Furthermore, this methodology has been shown to improve the design of the robot itself. Indeed, the process of understanding how astronauts recognize or do not recognize situations did not affect the OA system only, but suggested new devices on the robotic arm.

Acknowledgements

Thanks to Michel Maurette and Mario Delail for their excellent comments and advice. Nathalie Mathé helped us a great deal during the knowledge acquisition experiments, and she is currently very deeply involved in the continuation of this work. Thanks to Philippa Gander for critical comments on the manuscript. This work was supported by CNES contract 2 CDE 2625. NEXPERT OBJECT is a trade-mark of NEURON DATA.

References

Boy, G.A., (1986), "An Expert System for Fault Diagnosis in Orbital Refueling Operations", AIAA, 24th Aerospace Sciences Meeting, Jan. 6-9, Reno, Nevada.

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

Boy, G.A., & Nuss, N., (1988), "Knowledge acquisition by specialization/structuration: Application to Intelligent Tutoring Systems". Proceedings of the EuropeanWorkshop on Knowledge Acquisition, Bonn, June

Boy, G.A., & Delail, M., (1988), "Knowledge acquisition by specialization/structuration: A space telemanipulation application". Workshop on Integration of Knowledge Acquisition and Performance Systems - AAAI-88, Saint Paul - Minnesota - USA.

De Kleer, J., (1986), "An Assumption-Based TMS", Artificial Intelligence, 28, pp. 127-162.

Forgy, C.L., (1982), "RETE: A Fast Algorithm for the Many Pattern / Many Object Pattern Match Problem", Artificial Intelligence, 19, 1.

Laird, J.E., Newell, A., & Rosenbloom, P.S., (1984), "Towards chunking as a general mechanism", Proceedings AAAI-84, Austin, Texas.

Nagel, D., (1986), "Pilot of the future: Human or Computer ?", Workshop on Human-Machine Interaction and Artificial Intelligence in Aeronautics and Space, Toulouse, Oct. 13-14.

Rappaport, A.T., (1988), "Cognitive Primitives", Int. J. of Man-Machine Studies, 29, pp. 733-747.

Reason, J., (1986), "Decison Aids: Prothesis or Tools ?", Intelligent Decision Support in Process Environments, NATO Workshop, Nov. 11-14, Ispra, Italy.

Prospects for Conceptual Graphs in Acquisition Interfaces

Peter Eklund

Department of Computer and Information Science
University of Linköping
S-581 83 Sweden
pwe@ida.liu.se

John Kellett

Information Technology Research Institute
Brighton Polytechnic
BN2 4GJ, Brighton
United Kingdom
jmk@bton.ac.uk

ABSTRACT

This paper presents the results of applying several graph based interfaces as knowledge acquisition aids in Knowledge-based Systems (KBS) development and discusses various representations: decision trees, conceptual graphs and semantic networks.

The viability of a mapping from conceptual graphs to production rules as a means of eliciting knowledge is examined. A comparison is made between a production rule/semantic network representation and conceptual graph/semantic networks. Results indicate that the varying complexities of each representation ensure imbalanced acquisition between schema and heuristic elicitation.

To illustrate this, an example in the domain of engineering design for flight simulators is examined. On the basis of our experience we propose a minimal set of criteria for graphics based interfaces which must be satisfied for successful KBS prototyping.

1. Acquisition Methodology

Our approach to knowledge acquisition is that of developing knowledge engineering devices which capture the interest of the domain expert during prototyping. We accept that ideally the knowledge elicitation and representational phases of KBS development should be distinct[Hay87], but maintain that automated knowledge elicitation to develop definitive production systems is too ambitious (at least for our purposes) and instead develop interfaces as prototyping aids in engineering domains. In the KBS paradigm we discuss, we try to match the representational capabilities of the software with the expectations of experts for acquisition and elicitation, although we agree that there is a dominance relationship between the tools and expert, i.e., the expert compromises his knowledge to fit the tools[Van87].

Despite this, production rule (PR) acquisition serves to instruct the knowledge engineer who is seen as analogous to the protege of the expert. In the sense that the expert can also role-play at being a novice, a simply understood and usable interface is a pre-condition for role flexibity.

We concentrate on graphic knowledge representations because the domain experts in engineering design and project management have little difficulty relating to network representations. This is largely due to their strong socio-cultural familiarity with networks (e.g. PERT, CPA, etc) and in this way they match their expectations. Furthermore computer literacy is high in these fields and tools of this type do not alienate the expert as may be the case in less technical domains.

Further in support of this prototyping methodology for engineering domains is the observation that different experts sometimes agree with each others models but when they argue (as is more likely) the visual model improves the clarity of the discussion. Graphics based tools limit misinterpretation by allowing experts increased involvement with knowledge modeling. Arguments are often resolved by trying different approaches and testing there consequences.

Several modeling interfaces are discussed here and all are used in conjunction with a production rule based system called RBFS[Bar87] (Rule Based Frame System). VEGAN[Kel88] (Visual Editor for the Generation of Associative Networks) is used for describing semantic networks. KET[Esf88] (Knowledge Encoding Tool) organizes production rules into decision trees and CGT[Kuc89] (Conceptual Graph Tool) uses conceptual graphs as structures analogous to production rules.

2. Production Rules and Semantic Nets

RBFS is a frame based system which integrates three well known knowledge representation techniques: semantic networks (SN), production rules (PR) and frames. A domain can be incrementally described using these representations and the system will integrate them into an underlying frame structure. The novel feature of the system is termed *distributive inference* which is the ability to notify the inference engine about the rules which can reason about particular frames.

The accompanying development methodology concentrates on making the distinction between information and knowledge[Mar87]. Information represents a description of the world environment, the particular problem at hand, the evidence, and even the possible solutions. Knowledge forms the decision making capability transforming the evidence into conclusions. Conclusions themselves represent a particular manifestation of information transformed by knowledge. As a development environment and methodology RBFS

has proved successful in a number of diverse domains.

VEGAN is a directed graph editor tuned to the particular application of RBFS. It allows interaction with the information base via a graphical interface with the intent that the user need not have any contact with the underlying frame environment. An extended network syntax developed to enable a more expressive representation of the commonality between the rule and frame base. Nomenclatures to distinguish types of links between concepts translate into pointers to verify the completeness of the rule-base. These act as a key into the knowledge base and possibility, necessity and existential operators result. VEGAN also has an ability to filter the displayed information model. This reduces the complexity of the network presented to the user who is usually only interested in a small subset of the information displayed. An extension of this capability is that the information model can be used to visualize the system running. This feature is called *network animation* and VEGAN has facilities for the automatic placement of instantiations of specific concepts as they occur.

To determine the computer installation required;
IF 1. A proposal has been approved.
2. Company proposing has specified the aviation authority has fixed minimum requirements.
3. The aviation authority specifies the vision software should be at least of standard "1".
4. No vision software for proposals already approved have vision software level > "1".
THEN conclude that the vision software level for the simulator is "1".

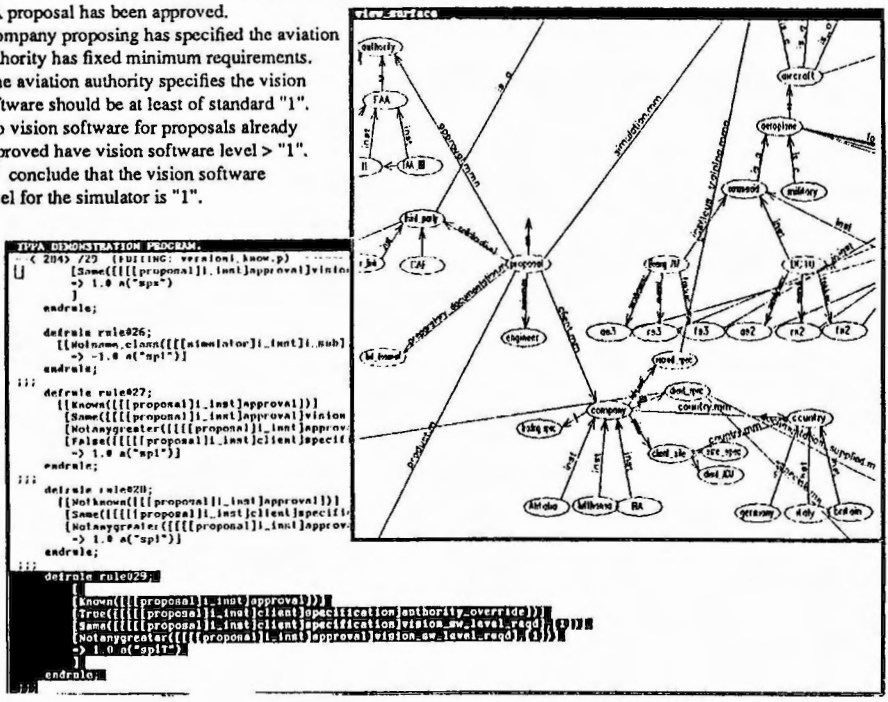


figure 1

Figure 1 shows a production "rule029" and the relevant hierarchy with which it deals.

What's noticeable is that the rule is rather difficult to read because premises specify just the base concept and then fan out via relational links. This is where the VEGAN representation helps. If we start at *[[proposal]i_inst]* we refer to all the instances of the class "proposal". Following the "approval" link we arrive at "authority" and discover something about the domain, that every proposal for a flight simulator needs approval from an aviation authority.

Similarly, for the second premise we see that a "client" is a "company" which has a "specification" called a "client_spec". A slot in the "client_spec" reads "authority_override", meaning that the recommendations of the aviation authority for vision software are enforced.

The third and forth premises say, for all the proposals find those with "vision_sw_level" of "1", make sure that this is consistent with the "vision_sw_level" specified in the "proposal" and conclude that each simulator that satisfies these conditions has "sp1T" instantiated.

We have now fully interpreted rule029, what did we use to do it?

- *an understanding of the domain.*

  The domain is known to be bid proposal in flight simulator construction. In order to interpret the rule we had to understand what the symbols "authority", "proposal", "client", "vision_system", "company" and "client_spec" represented. All reflect the vocabulary of the domain expert and should also be known to the knowledge engineer.

- *why the conclusion is being made.*

  Knowledge of the software level helps cost the proposal, a less sophisticated software vision system makes for cheaper hardware and a less expensive simulator. Once again this "goal" should form part of the shared knowledge of both expert and knowledge engineer.

- *semantic network schema.*

  In order to fully read the paths used in the production rule the VEGAN representation was utilized. It revealed what links led to what nodes and showed how the object schema was structured. Lack of familiarity with semantic networks does not prohibit expert understanding.

- *syntax and morphology.*

  At a more detailed level we had to read around the syntax of the production rule format and decide how to read the premises and the rule conclusions. Even if one knows nothing of backward chaining production rules, because the rule was in a text format we naturally assumed to read it like normal language. Top to bottom and left to right seemed natural and proved correct.

3. Production Rules and Decision Trees

KET was designed to provide the expert with access to the rule base and a soft landing from the syntax of production rule specification. It is a folding editor which reveals the relationships between production rules in the knowledge base. A decision tree results (see figure 2). Not only is the intention to reduce direct contact with the production rule syntax but to also reveal the associations between intermediate conclusions drawn.

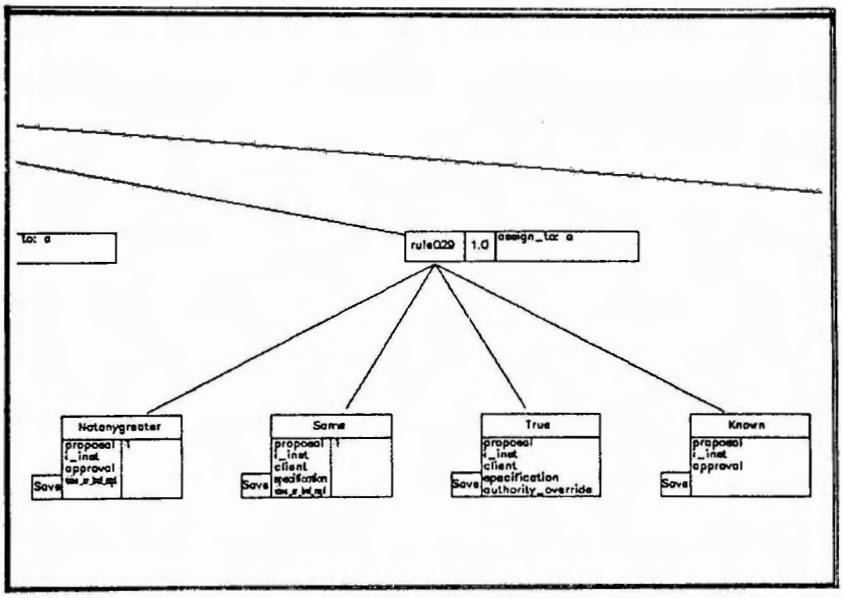


figure 2

The hierarchical structure, with the trunk node to the top and the leaves to the bottom of the view surface reveals the structure of rule associations. Such a representation is probably better expressed with leaves to the left and trunk to the right. This not only suggests flow or direction of reasoning more readily but also avoids confusion between the decision tree representation and the object-relation hierarchy. For example, in figure 2 the four premises of rule029 are not of the super/subordinate class variety but express a type of dependency relationship between the symbol rule029 and the premises which comprise it.

Figure 3 shows an improved prototype. Here the leaves are to the left and the trunk to the right. In this representation the different three-dimensional shapes express subcomponents of the rule base, i.e. premises, rules, functions, questions and data. The degree and type of shading indicate the supporting certainty (front solid) or uncertainty (rear shaded) of the various knowledge base elements.

The left to right representation naturally indicates the flow of decisions in the knowledge base. Primative facts to the left and a refined higher level conclusions to the right. Combination and decomposition can be easily ascertained. There is no suggestion of taxonomy, the representation being clearly distinguishable from an "is_a" hierarchy.

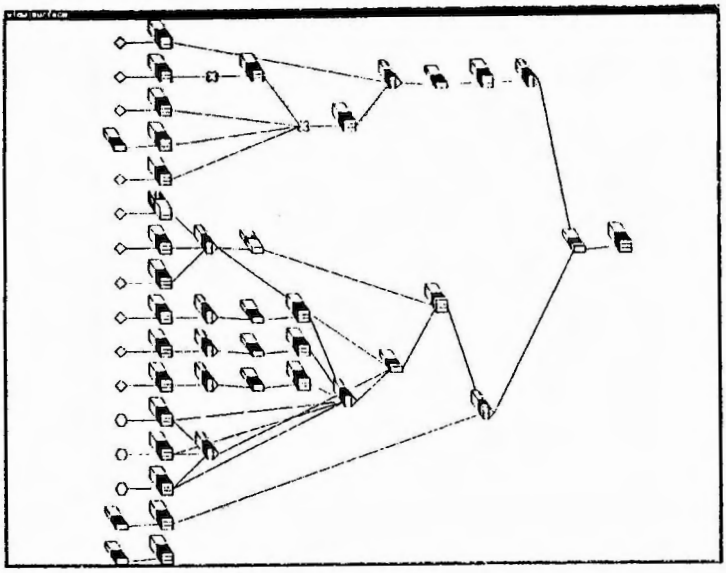


figure 3

4. Acquisition using Semantic Nets and Decision Trees

The results of the KET/VEGAN combination demonstrate a tendency to construct qualitatively better models in well defined domains simply because the expert becomes involved to a greater degree in specifying and debugging the model.

The interfaces also allow the expert to work independently of the knowledge engineer for substantial parts of the system development, the role of the knowledge engineer being limited to initially explaining VEGAN/KET's functions and providing occasional guidance in the development of the modeling schema. Given this impetus, VEGAN/KET improves the efficiency and speed at which domain models can be developed because the expert can experiment with different models and is encouraged to prototype. The result is that the domain theory can be quickly spanned. That the methodology and tools would achieve similar success in less technical domains is unknown.

5. Representational and Ergonomic Deficiencies

Throughout the development of projects we noticed that inexperienced knowledge engineers, and all the experts we dealt with, tended to place an unnecessary emphasis on building and modifying the semantic network. Often this has occurred at the expense of production rule definition, even on occasions when easier solutions were at hand through re-coding rules. There are two explanations for this phenomenon;

- *the rule encoding interface lacked intuitive appeal.*

  KET redresses this imbalance by providing a graphics map of the production rule knowledge-base and an interface which is more or less identical to VEGAN.

- *production rule coding is a fundamentally more complex task.*

  For the expert, identifying object hierarchies and their properties is easier than codifying heuristic knowledge. Still, without recourse to the psychological complexity of heuristic identification[Cle87] and based on the assumption that the expert is reformulating knowledge into a PR format, the ergonomic imbalance remains unsolved due to the ease in software by which object-relation hierarchies can be constructed in comparison with production rules. Also given that a PR representation is insufficient for describing some domains suggests the need for a more sophisticated interface for rule definition than for describing schemata.

6. Using Conceptual Graphs as Productions

Conceptual graphs[Sow84] model the underlying semantics of a knowledge-base and provide a common language or framework for the development of knowledge-base systems. Using the CGT, conceptual graphs are used for describing the logical dependencies inherent in a particular domain. Semantic networks are used in conjunction with conceptual graphs for defining the object-relation schema.

In the example based on figure 1 the reader may have noticed that there was no mention of the fact that both the "i_inst" link and the "authority_override" where difficult, if not impossible, to locate in the SN representation.

The clarity of the links in the SN are determined by the spatial distance and orientation of the nodes and the links between them. It's easy to write a simple fix procedure to locate nodes an appropriate distance from one another so that the text is readable but, by doing so, large networks may become so diffuse that the network is too sparse to be of help in deciphering the rule. Moreover, the spatial orientation of links and nodes is the responsibility of the expert or knowledge engineer. Client placement helps locate relevant parts of the network, automatic placement diminishes the ability to recognize object clusters.

The point to note with reference to figure 1 is that what cannot be read in the production rule is readily available in the network representation and *vice versa*. The deficiencies of one representation are the advantages of the other.

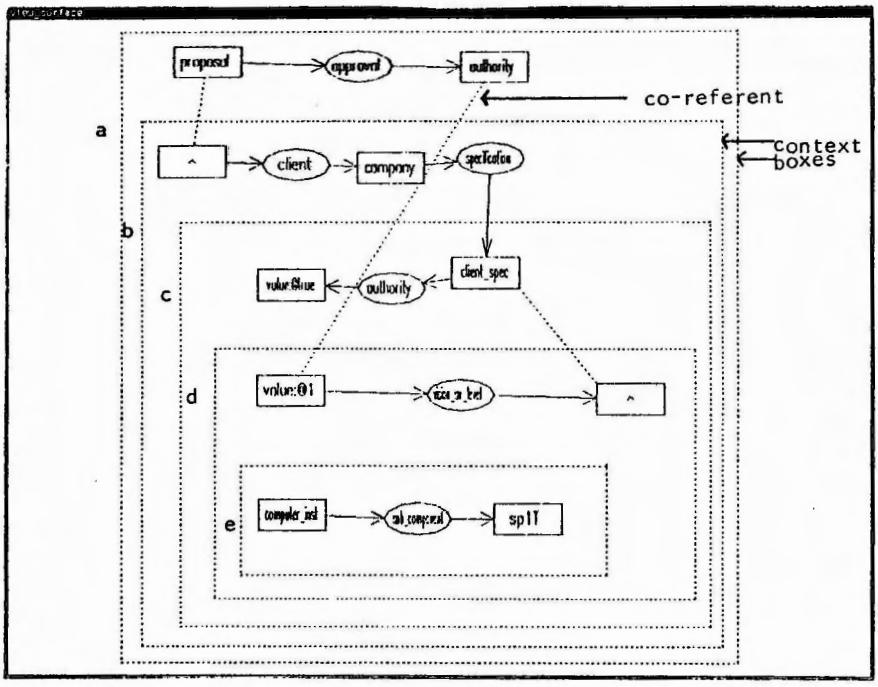


figure 4

6.1. Expressiveness of the Representation.

Figure 4 shows the conceptual graph which is equivalent to rule029. The improvements in readability are noted as;

- *nodes as well as relational links are specified.*

  In the production rule representation this was not the case. This means we need only refer to the SN when we need to know about the structure of the database rather than to correctly identify paths used by a rule.

- *predicates in a PR premiss are implicit.*

  Figure 4 shows that the predicates "Same", "True" and "Known" etc, are no longer required. This is achieved by including tests for literals at the end of paths, i.e. slot values are revealed in the representation.

- *multivariable unification via "coreferent" links (see figure 4).*

  Multiple slot values at the end of search paths need not be dealt with by virtue of coreferent links. The fourth premiss used in rule029 in figure 1 is therefore made redundant in figure 4.

Although the benefits of the representation are significant, we also note some disadvantages.

- *readability*

  For the sake of consistent graphics handling, as well as aesthetics, both concepts and relations are a constant shape and size. Text in relational links like

"vision_sw_level" are now unreadable in both representations. Formerly, one could rely on the PR for a description of the relation names.

- *are context boxes intuitive?*

  Context boxes are negated (i.e. dotted) because the proof mechanism operates on the same principle as a Prolog interpreter i.e. we want to prove the assertions made in the CG by defining goals which are negations of the assertions and proving them false. This can be difficult to explain to the expert, encouraging him to define his knowledge in terms of its negation often seems counter-intuitive. This could be disguised easily enough, but would represent a compromise of the formal theory by obscuring the inference process.

- *labeling the conceptual graph.*

  The CG in figure 4 is analogous to rule029, should it be labeled as such? We experimented with a number of labeling schemes. These included;

  - positional orientation schemes, e.g., *"r5c4"* would indicate the CG in the fifth row and forth column with the origin at the upper left.
  - Most recently defined e.g. *"CG5"*, the fifth CG presented to the system by the expert/KE.
  - Symbolic labeling with respect to the goal e.g. in figure 4 *"computer-sub-component-1"*.

  The fact that no single labeling scheme is clearly preferred suggests that multiple planes of the CG base should be defined. The client could then ask for the representation which suited; i.e. show me clusters of the CG base ordered by the conclusions they draw; order the CG base by chronological definition; order by goal precedence (highest level goal → lowest); order by certainty factor and so on. The problem with this approach is that clients loose spatial identification with where on the view surface they have defined CGs.

- *representing certainty factors.*

  Figure 4 presents little in the way of information about certainty factors since it is known with certainty 1.0 to be true. Representing certainty is problematic, since there is some doubt as to where the certainty factor (CF) attaches in the graph. In terms of this implementation the CF belongs to the entire CG since it is used for resolving the firing precedence in the conflict set. From an ergonomic view it seems natural to attach the CF to the innermost context and this helps distinguish the conclusion from the antecedents. In the implementation a rule fires when the premise path it evaluates combines to certainty ≥0.2. For the sake of consistency, each antecedent context should likewise reveal this. Specific instantiations of paths for which the rule is a generalization may have relations of varying certainty and therefore the CF displayed in context boxes only has meaning as a CG is firing. This raises questions of the relevance of such a trace for explanation and debugging.

6.2. Spatial Considerations

- *editing conceptual graphs*

  Adding a new context to an existing conceptual graph requires major reorganization of the positional orientation of existing contexts. In the PR representation, the addition of one or more new premisses is a trivial insertion using a text editor. In the CGT it becomes a major chore. It could be argued that this is the result of the

deficiencies of our prototype but in our view it is further support for a reversion to the textual form for conceptual graphs. We observed that experts and KEs who used the CGT were reluctant to modify CGs and would search for fixes. In one extreme case the KE coded around just such a problem simply because adding new contexts to several CGs would take too long.

- *inference direction.*

  The graph can be read either from outer context box **a** towards the context box **e** or out from **e** to **a**. Is the CG a backward or forward chaining rule? In conceptual graph theory CGs are meant to represent purely declarative knowledge. Figure 4 as a backward chaining rule on the basis of the structure of context boxes violates this no function in structure principle which the formalism advocates. That the CG has to be interpreted as a forward or backward chainer in the first place is pre-requisite on delivering a computationally feasible implementation. In the production rule the "=>" can be construed as a synonym for implication $\rightarrow$ in first order logic. We therefore distinguish the first three lines of the rule as conditions which should be satisfied in order for the final line of the production to hold. The LHS and RHS are distinct in the PR form.

7. Acquisition using Conceptual Graphs

The assertion that conceptual graphs could be used as analogous to production rules seemed obvious since we were familiar with the different representations. We neglected the fact that this familiarization was a progressive and non-parallel process. We first gained considerable experience with production rules and semantic networks before realizing that not everything could be adequately modeled using them. When realizing this, we began using conceptual graphs as a means to overcome more complex representations. The question would be whether the naive knowledge engineer or expert could make the distinction as easily? Failure to ask ourselves this question meant that we were frequently surprised at the difficulties that were experienced by our clients.

8. Criteria for Successful Interfaces

Finally, although our experience does not represent a complete survey of all graphics representations likely to be of use in KBS development, we suggest the minimum criteria for the successful implementation of such representations are;

- *socio-cultural identification of expert.*
- *reasonable cognitive basis.*
- *sound theory of inference and combination.*
- *spatial orientation which reveals the structure and intent of knowledge.*
- *graphics which clearly distinguish and minimizes the number of objects, portrays attribute and dimensional characteristics.*
- *encourages of completeness and consistency.*
- *clear representation and handling of uncertainty.*
- *readability and modifibility.*

9. Conclusions

The notion of integrating a number of well known graphical knowledge representation techniques into a unifying approach, based on semantic networks and conceptual graphs, has significant ergonomic appeal, the idea being that the human-computer interface is balanced for both schema and heuristic definition and neither representation suffers at the hands of software ergonomics.

Our experience indicates that although the dualism provides an attractive human-computer interface for the experienced knowledge engineer, the expert is often left confounded by the possibilities presented. If this is the case then all that has been done is to provide another level of abstraction on an already crowded workbench. It could even be argued that the experienced knowledge engineer could just as easily have used a production rule representation.

Our experience suggest that a semantic network editor in combination with a textual heuristic representation and a decision tree as a rule-based road map is a more natural acquisition aid being understood by both expert and Knowledge Engineer.

Acknowledgement

The interfaces discussed in this paper represent a summary of developments at *Information Technology Research Institute Brighton Polytechnic UK* from July 86 to 88. Our thanks to our colleagues at *ITRI* and *IDA* and particularly Jonas Löwgren for the discussions from which this paper resulted.

References

Bar87.T Barber, G Marshall, and J Boardman, "A Philosophy and Architecture for a Rule Based Frame System (RBFS)," *Int. Journal of AI in Engineering*, 1987.

Cle87.D Cleaves, "Cognitive Biases and Corrective Techniques: Proposals for Improving Elicitation Procedures for Knowledge-Based Systems," *Journ. Man-Machine Studies*, vol. 27, pp. 155-166, 1987.

Esf88.L Esfahani, "A Self Organising Rule Elicitor," *2nd European Workshop on Knowledge Acquisition for KBS*, Bonne, 1988.

Hay87.
A Hayward, B Wielinga, and J Breuker, "Structured Analysis of Knowledge," *Journ. Man-Machine studies*, vol. 26, pp. 487-498, 1987.

Kel88.J Kellett, "VEGAN - A Visual Editor for the Generation of Associative Networks," *Int. Journ. Knowledge Based Systems*, 1988.

Kuc89.
P Kuczora and P Eklund, "A Conceptual Graph Implementation for Knowledge Engineering," *in Human and Organisation Issues in Expert Systems (ed A Hart & D Berry)*, Krogan Page , 1989.

Mar87.
G Marshall, T Barber, and J Boardman, "Methodology for Modelling a Project Management Control Environment," *IEE*, vol. 134-D, no. 4, pp. 287-300, 1987.

Sow84.
J Sowa, *Conceptual Structures - Information Processing in Mind and Machine*, Addison-Wesley, Reading, Massachusetts, 1984.

Van87.
F Vandamme, "Knowledge extraction from experts in view of the construction of expert systems," *NATO ASI Series*, vol. f35, pp. 321-337, Springer-verlag, Berlin, 1987. in Expert judgment and expert system (ed. J. Mumpower et. al.)

Knowledge Acquisition in the Domain of CNC[1] Machining Centers: the MOLTKE Approach[2]

K.-D. Althoff, S. Kockskämper,
R. Traphöner, W. Wernicke
University of Kaiserslautern
Dept. of Computer Science
P.O. Box 3049, D-6750 Kaiserslautern
West Germany

B. Faupel
Technical Universty of Aachen
WZL
Chair for Metrology and Quality Assurance in Production
Steinbachstr. 53B, D-5100 Aachen
West Germany

Abstract

MOLTKE is a research project dealing with a complex technical application. After describing the domain of CNC machining centers and the applied KA methods, we summarize the concrete KA problems which we have to handle. Then we describe a KA mechanism which supports an engineer in developing a diagnosis system. In chapter 6 we introduce learning techniques operating on diagnostic cases and domain knowledge for improving the diagnostic procedure of MOLTKE. In the last section of this chapter we outline some essential aspects of organizational knowledge which is heavily applied by engineers for analysing such technical systems (*Qualitative Engineering*). Finally we give a short overview of the actual state of realization and our future plans.

1. Introduction

The aim of the MOLTKE[3] -project is to show that a complex technical problem can be put to an economical solution using expert system technology. The domain of CNC machining centers is used as an exemplary domain. The solution encompasses the diagnosis of the CNC machine as well as the adaptation of such a diagnosis system to the further development of the machining center by the manufacturer.
The complexity of this problem requires rather an integration of different methods than straight forward knowledge acquisition (KA) methods. Firstly, an engineer has to interpret the construction plans of the machine and to interview the domain experts of the machine manufacturer. Secondly, a software tool has to be applied that enables the engineer to implement the diagnosis system using his own concepts and without having to know too much about the programming system. Thirdly, a natural knowledge representation is needed which can make the acquisition of the involved domain heuristics as easy as possible.

2. The Domain of CNC Machining Centers

Today, the development of expert systems for diagnosis problems has a high bearing on mechanical engineering. The motivation for the diagnosis of a CNC machining center is based on the requirement of increasing the reliability and applicability of complex machines. These depend on the expenditure for repair and service. The expectation of short duration for detecting and removing actual faults of machining centers can only be satisfied by using qualified service personnel with special education. The real situation in industry shows that the potential of appropriate experts is limited. It is expected to have instruments supporting diagnosis. The difficulty of fault diagnosis arises from the complex technical realization of these machines; the behaviour and function, of course, rests on the coordination of different machine components (electric, electronic, mechanical, hydraulic and pneumatic). Experts like service technicians from the machine manufacturer arrange this knowledge. Therefore, fault diagnosis can only be accomplished if it is possible to evaluate information about the

---

[1]) CNC = Computerized Numeric Control

[2]) The work presented herein was partially supported by the Deutsche Forschungsgemeinschaft, Sonderforschungsbereich 314, project X6 MOLTKE

[3]) Models, Learning and Temporal Knowledge in an Expert System for Technical Diagnosis

function of the machining center and its components. The task of building a fault diagnosis system requires a systematic analysis of the following knowledge classes:

1. Special technical knowledge (deep knowledge) and
2. Special heuristic knowledge (surface knowledge).

The characteristic components of machining centers are axes, spindle, tool and pallet changer and the control unit (CNC). The function of all components is realized with electric and mechanical elements. The specific knowledge about the machine components and different elements deals with:

- local features that show how these components work and
- the connections between the particular systems, which describe the global function of specific machine working cycles. The I/O-signals of sensors and components are important sources for the status of the connection ports.

Other kinds of technical knowledge are fault probabilities. One field of quality assurance is the analysis of weak points and the life-span of specific machine elements. One can draw conclusions from these results about those elements having a higher probability in diverse fault situations.
The geometrical accuracy of workpieces is very important in the manufacturing process. The cause for product deviations originates from the factors implied in process. The starting-point for the diagnosis of production faults are the data from special measuring helps (coordinate measuring machine, gauge, etc.). Such technical knowledge requires a deep understanding of the different involved subjects: control engineering, (mechanical) machine elements etc.
The significance of heuristic knowledge is shown by way of human experts solving the given fault problems, e.g. which strategies and abilities they use to recognize the relationship between different information. It is the quick and sure apprehension of apparently independent data which is a characteristic of experts. In (nearly) all fault situations they have a lot of similar fault examples (this leads us to the introduction of diagnostic cases in the next chapter).
The quality of a diagnosis system is essentially influenced by the appropriated information sources (electric and mechanical design plans, fault probabilities, machine and system manuals, motivated and talkative experts) and - that is decisive - the ability of structuring, connecting and evaluating all acquired details being used for building a knowledge base.

3. Applied Indirect Knowledge Acquisition Methods

The machining center we use for diagnosis (Maho MC600) is available in the mechanical engineering institute at the WZL Aachen. Therefore a significant amount of know-how concerning the functional behavior of the machine can be acquired from our cooperating engineer. Additionally, all the following information was evaluated for the realization of the knowledge base of MOLTKE:

Specific technical knowledge
- electric design plans
- mechanical design plans
- hydraulic design plans
- pneumatic design plans
- statistic quality assurence reports
- fault probabilities
- machining manuals
- measurement manuals
- error messages

Human experience knowledge
- observations of the service technician
- communication and discussions with other service personnel

Our engineer had the opportunity to take part in a special training program of the manufacturer during which he could interview expert sevice personnel. Service reports and plans of similar machines were obtained. Many other companies have been visited to get information both about the causes of deviation of the geometry of workpieces and the structure of the control unit. KA methods which have been applied are the result of studying all the mentioned source material and additional unstructured interviews with expert service personnel. For the acquisition of the heuristic knowledge, *diagnostic cases* have been introduced to "mirror" concrete diagnostic situations. They consist of a machine fault and all the symptoms which have been acquired by the respective service technician.

Such a natural knowledge representation is a minimal requirement, because of the KA problems described in the next chapter.

4. Knowledge Acquisition Problems

A main point for the evaluation of the specific technical knowledge are the already existing possibilities of the CNC for finding machine failures. In most cases the control unit provides an error message, if a malfunction during the machine cycles or an emergency stop can be noticed. The CNC generates warnings without stopping the manufacturing process, if no severe disturbances occur. One problem is the varying information content of the error messages. In some cases the fault causes can directly be read from the error statement, in other situations the message is only the starting point for the diagnosis process. The error messages are divided into two main fault classes [Pfeifer, Held, Faupel88]:

CNC fault message	PLC[1] fault message
system errors	I/O-errors
programm errors	axes- and spindle errors
data errors	
working errors	

The condition for creating PLC error messages depends on the observation of the temporal alternation of the I/O-signals in comparison with the correct function of the machine cycles. It is not quite trivial to select a diagnosis strategy based on an occurence of an error message only. First of all the temporal sequences of cycles of the machine parts have to be known. In general the service personnel looks for the machine status at the point of time when the error was determined. The machine part cycles can be described simply by the following sequence:

1. input signal
2. controlling different elements (switches, relays, valves, ...)
3. feedback impulse to the CNC

The exact correlation between the design and the realization of the machine cycles has to be obtained from several plans and diagrams. Thus the control information for individual elements can be acquired from the electric, hydraulic and mechanical design plans. The sequence of the part cycles cannot yet be derived from such plans. The access to this very important information about the function and behavior of the machine may be possible by testing the function of a real machining center or by extracting the details from complex PLC-programs. In general it is problematic to get this secret information belonging to the protected know-how of software companies.
Another problem of KA is the optimization of problem solving steps and strategies as a function of different fault situations. Appropriate strategies and steps of the diagnostic procedure are determined by the different goals that serve as a guideline for the respective service technician, like the orientation according to fault probabilities, the experience, the difficulty of testing and measuring, and temporal estimations of the expenditure of testing and measuring. All relevant facts have to be considered for finding optimized diagnosis steps. Often the fault probability suggests errors of electric switch elements, but in the case of hidden switches the necessary tests are too expensive. Usually less probable fault causes are verified in such cases. The known fault probability reflects the analysis of all types of machining centers. Characteristic weak points of particular machines cannot be derived from global statistic informations. An important problem is the classification of fault symptoms by degree of complexity:

difficult	easy
noise and vibration derivation	error message
smell derivation	state of electric and mechanical
temperature derivation	elements

The correlation of "difficult" symptoms to possible fault patterns is influenced by subjective decisions, i.e. a noted deviation in the normal noise level can only be used for diagnosis, if a

[1]) PCL = Programmable Logic Control

connection to the actual fault situation exists and - that's important - the noise impression can be combined with machine components.
The expenditure for acquiring the expert knowledge differs with the different subjects. The analysis of all information about electric components is extremely relevant for the diagnosis of our domain. The KA of mechanical sequences is essentially difficult and also unfit for diagnosis. The exchange of mechanical elements like the spindle bearing lasts more time than any exchange of electrical components. A further problem is the description of the necessary qualification and test methods for analysing mechanical faults. The minimal requirement the domain experts have to meet is to know all about the function and behaviour of CNC machining centers, as this is the central part of their education by the manufacturer. The real experience of these experts is developed (only) during their practical work as service technicians. The main problem of the KA is the identification of such heuristic knowledge.
The intention of the consultation of the expert is at first focused on the registration of the global methods that the expert uses in most fault situations. The generalization of the global methods leads to a systematic procedure of the diagnosis process. The next step is the analysis of the diagnostic procedure used in special cases. The exclusive experience, i.e. the intuition of unconventional handling of diagnosis steps, is only obvious in difficult diagnosis problems. By consulting several experts the own observation impresses different methods and strategies for diagnosis. Similar to other domains, different opinions of experts are usual. The possibility for an outsider knowledge engineer to decide between several expert opinions is still an unsolved problem. Finally, the KA with the "use" of a knowledge engineer is an iterative process for optimizing and completing the knowledge base.

5. Towards a more Direct Knowledge Acquisition Mechanism

To understand the ideas and the mechanisms applied in the KA tool we will describe, it is important to examine the underlying structures of the knowledge. Therefore we first give a closer look to the knowledge representation in MOLTKE.

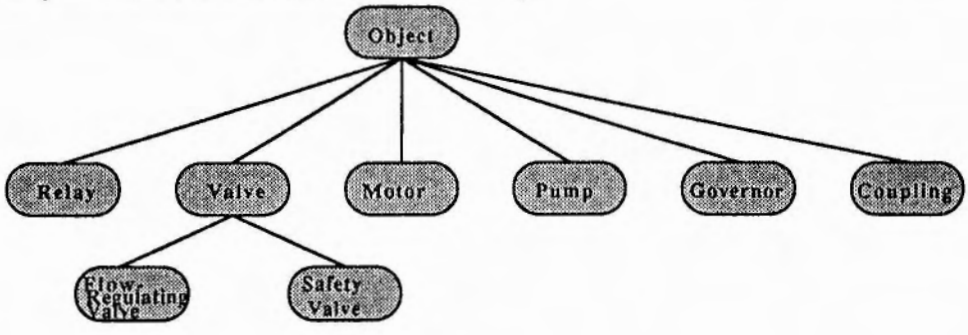
5.1. Knowledge Representation

In the domain of CNC machining centers, we have to deal with large knowledge bases. In order to organize and manage them, it seems necessary to split up the knowledge vertically as well as horizontally.

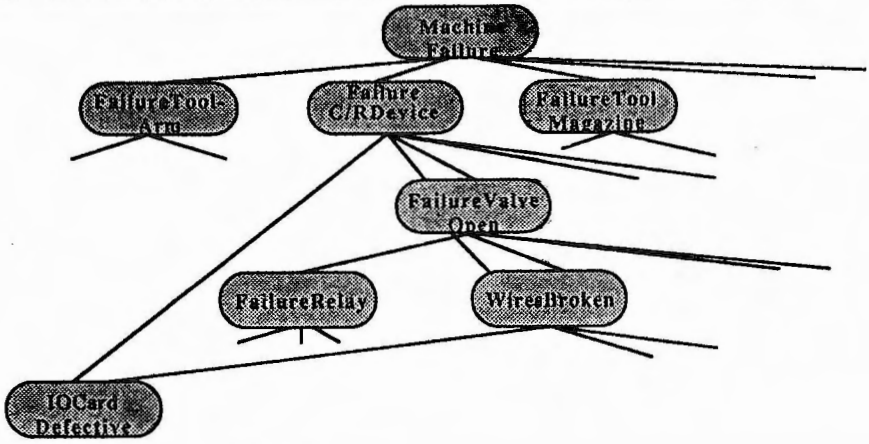
5.1.1 Vertical Modularization

In MOLTKE there are two kinds of taxonomy:

Component Hierarchy - Every component of the CNC-machine is represented as a frame. The components are structured hierarchically by is-a relations. Every component frame describes an abstract component that has the typical features of some part of the machine. Depending on the level of specificity, this hierarchy can extend over several layers. Concrete components are instances of component frames at the lowest level of this hierarchy.



Context Heterarchy - Besides this component hierarchy there exists a diagnosis heterarchy, in MOLTKE called context heterarchy. The knowledge in MOLTKE is organized on different levels of abstraction. This corresponds to the vertical modularization. The resulting modules are called contexts. Each context represents a diagnosis, which leads to more specific diagnoses represented by its subcontexts. Contexts are connected with their subcontexts by refinement links.



5.1.2 Horizontal Modularization

The knowledge represented in each context is subdivided in the following categories, which are explicitly and seperately represented.

Control Knowledge - Control knowledge is represented, as already mentioned, seperately from domain-specific knowledge. This is of decisive advantage for the development of large knowledge bases, because an application-oriented system often requires problem-specific control strategies. For this reason there is a need for a formalism that allows a simple description and modification of control strategies. In MOLTKE we use control rules to describe the proceeding of the system.

Knowledge about fault situations - In MOLTKE facts about a fault are represented as preconditions of contexts. These preconditions are declarative descriptions of the symptomatic pattern of a fault. In a given situation it is easy to check in the subcontexts, whether there is a fault or a reference to a fault. This is useful if there is data available that is not directly requested by the system, but sporadically supplied by sensors. Furthermore, the explicit representation of faults makes it possible to verify hypotheses of the user and allows an easier integration of a case-based mechanism, because diagnostic cases have a similar description.

Knowledge about Order - This kind of knowledge is necessary to answer the question: "Which symptom must be asked next ?". It contains a description of how to prove a fault, i.e. how a situation can be reached as quickly as possible, in which a fault can be verified. Knowledge about order is not necessary to choose a special test, but to choose which symptom is to be asked next. Possibly there are several tests for a symptom which differ in cost and time. The symptom itself determines which test is favourable at a given moment.

Determinations - Determinations represent correlations between symptom values. They can be either of functional or of empirical nature (for a formal definition see also 6.1). Functional correlations (total determinations) between symptom values can be taken from construction plans of the machine or can be generated automatically out of a model of the machine [Rehbold89]. Total determinations correspond to constraints between symptom values. They need to be used in one way (like rules), because in the diagnosis a defect in the machine is assumed. Partial determinations can be generated from diagnostic cases on the basis of empirical data.

An essential part of the knowledge base of MOLTKE is represented in the form of rules and formulas. For an efficient treatment of that knowledge a RETE-similar network is built. This network is used to evaluate the preconditions and rules of the contexts in an event-driven manner. The basis for evaluation is a 3-valued logic, that simplifies, among other things, the treating of incomplete knowledge.

5.1.3. Knowledge Acquisition

KA in MOLTKE is simplified by separating the different kinds of knowledge. The facts about the symptomatic of a fault is relatively easy to obtain from the expert. It is much more difficult to acquire the heuristic knowledge: "Which action must be done next to reach a diagnosis ?".
The following principles must be taken under consideration when ascertaining symptoms:

- minimization of cost and time
- concentration on the most relevant symptoms
- avoidance of a confusing order when ascertaining symptoms
- priority treatment of directives from the user

Often a diagnosis can be reached in different ways. The way which leads to a diagnosis can differ in cost and time of the tests which must be carried out. The decision of which way to choose depends on many criteria:

- educational background of the operating and service personnel
- experience of the operating and service personnel
- concrete diagnosis situation
- cost, time and accuracy of the tests
- frequency of symptoms/faults
- urgency of a diagnosis

There are six possible strategies resulting from these factors:

- importance-oriented strategy for choosing symptoms (i.e. symptoms which are relevant to detect a failure of a part of the machine)
- frequency-oriented strategy (statistical data about failures of a part of the machine)
- cost/time-oriented strategy (i.e. cost/time of the test which must be carried out)
- difficulty-oriented strategy (i.e. difficulty of the test which must be carried out)
- derivation-oriented strategy (avoiding useless questions to the user, because this strategy prefers symptoms which can determine other symptom values)
- individual strategy (depending on the experience of the user)

As the table shows, the proceeding during the diagnosis process is highly dependent on the qualification of the diagnostic staff, e.g. many tests are not allowed to the machine operator. It is important for the acceptance of the system, that it is flexible with regard to its proceeding, i.e. it must not pursue only one of these strategies, but depending on the kind and qualification of the user, it must put different strategies at the user´s disposal and allow him, at any time, to check the symptoms he wants to.

Requirements:

- realization of different strategies
- consideration of user´s directives
- possibility to combine different strategies

Realization:

a. According to the object-oriented paradigm a strategy object is defined for each of these strategies (strategy objects define a partial order on symptoms/tests). Every context receives a strategy object which is equipped with a default strategy before an exact strategy is specified. This default strategy can be set by the user at the beginning of the session or can be fixed by the system. Furthermore, the strategy object contains information about the number of symptoms that appear in a context, as well as details about the number of symptoms that fulfill a certain criterion (these slots are set up automatically by if-added demons).
b. There are ordering rules to establish exceptions from standard strategies (but also for individual strategies, e.g. proceeding in accordance to machine cycles).
c. There is a menu, available at any time and containing all unknown symptoms, that allows the user to determine the symptom he wants to.
d. The combination of different strategies is realized by the introduction of connection operators.
e. The combination of strategies and rules is also realized by connection operators.

tests \ qualification level	1	2	3
opt. testing easy to recognize	D5 T5	D2 T1-2	D1 T1
opt. testing parts of the machine: relay, valves, motor,IO-state	D- T-	D4 T2	D1 T1
electr. testing resistance, stress, current	D- T-	D4 T1	D2 T1
hydr. testing components: pumps, valves, manometer, pipings, seals, flow-regulating valves	D- T-	D4 T3	D1 T2
pneum. testing tool holding fixture, worktable, access door, governors	D- T-	D4 T3	D1 T2
noise identification, interpretation	D10 T-	D4 T-	D5 T-
smell identification, interpretation	D10 T-	D7 T-	D5 T-
feeling identification, interpretation	D10 T-	D7 T-	D5 T-
control technology (metrology) governors, drive system, motor, measuring system,sensors	D- T-	D8 T5	D3 T4
control technology (CNC-technique) program technique, testprograms	D- T-	D8 T5	D3 T5
mech. testing gearing, guidances, bearings, couplings shafts,studs, shutters	D10 T5	D8 T5	D3 T5
methodology used instruments, methods	D- T-	D8 T5	D5 T3

Explanation:
qualification 1 --> machine operator
qualification 2 --> maintenance personnel
qualification 3 --> service technician
D = difficulty valuation: 1-10 - : not given
1 = very easy, 2-3 = easy, 4-6 = less difficult, 6-8 = difficult, 9-10 = very difficult
T = time valuation: 1- 5 - : not given
1 = very short, 2 = short, 3 = less long, 4 = long, 5 = very long

The determination of order when ascertaining symptoms can be done principally in three ways:

a. Establishing the order on the basis of syntactical criteria
b. Establishing it based on semantical criteria
c. Combination of syntactical and semantical criteria

Often there is much information missing, which is important for a useful proceeding in finding a diagnosis. That is the main problem, which arises by the incremental building of the knowledge base.

To examine this difficulty, we make a subdivision in four phases as a basis for building the knowledge base. Each phase corresponds to a certain level of knowledge of the knowledge engineer. According to this, the demands on the system are as follows:

- Possibility of incremental input of faults and symptoms
  There are two tools at the user's disposal: FrameBrowser and RuleBrowser. The RuleBrowser is a tool for the input of rules and formulas as well as for their management and organization. The FrameBrowser serves for the treatment of the definition of symptoms, tests and faults. Both tools were created on the model of the SystemBrowser of Smalltalk-80, i.e. their use corresponds to the usual Smalltalk philosophy.
- Mechanism to determine the proceeding in finding a fault (i.e. the system is able to run, even if the syntactic and semantic criteria are incomplete)

We give now a simple example from the domain of CNC-machines as an illustration of the individual phases. Our example shows only some details of the complex frame-structure.

5.2 A Demonstration of the Mechanism

5.2.1. Declarative Phase

Which objects do exist ? - In this phase a description of faults (contexts) is carried out on a relative high level of abstraction (in the sense of simple diagnosis) and with it a description of the symptoms and tests.

context: MachineFailure	
precondition:	true
symptoms:	(ErrorCode, IN34)
refinements:	(FailureClamping/ReleaseDevice, FailureToolArm)
diagnosis:	MachineFailure
strategy:	default

context: FailureClamping/ReleaseDevice	
precondition:	(ErrorCode = I59)
symptoms:	(IN36,OUT7,Valve5Y1, OUT24,Valve5Y2,Wires, IN32)
diagnosis:	FailureClamping/ReleaseDevice
strategy:	default

Definition of the symptoms, appeared in this context:

Definition of the tests:

symptom: ErrorCode	
type:	(I47, I67, I59)
importance:	very important
possible tests:	ErrorCode-check

test: ErrorCode-check	
precondition:	qualification 1
output:	Please check the error code !

symptom: IN34	
type:	(logical 0, logical1)
importance:	important
possible tests:	IOCard-check

test: IOCard-check	
precondition:	qualification 2
output:	Please check the IO-state < > !

symptom: IN36	
type:	(logical 0, logical 1)
possible tests:	IOCard-check
importance:	important
symptomGroup:	(OUT7, OUT24, IN32)

test: IOCard-Check	
precondition:	qualification 2
output:	Please check the IO-state < > !
difficulty:	qualification 2 -> easy to acquire qualification 3 -> very easy to acquire

symptom: Valve5Y1	
type:	(switch-position 0, switch-position 1)
possible tests:	Valve-Check
symptomGroup:	Valve5Y2

test: Valve-Check	
precondition:	qualification 2
output:	Please check the switch state of the valve <>
difficulty:	qualification 2 -> less difficult to acquire qualification 3 -> very easy to acquire

symptom: Wires	
type:	(ok, broken)
possible tests:	Wire-Resistance-Check

test: Wire-Resistance-Check	
precondition:	qualification 2
output:	Please check the
difficulty:	qualification 2 -> less difficult to acquire qualification 3 -> very easy to acquire

All other symptoms are defined in the same way.

A diagnosis run in this phase would lead to the following order when ascertaining symptoms (under the assumption that the importance-oriented strategy is the default strategy):

Output: Please check the error code !
Input: I59

After the input I59 the context FailureClamping/ReleaseDevice is reached. The default strategy does not lead us anywhere, because there is no information about the importance of symptoms. Because of the existing details about difficulties of tests, it is switched to the difficulty-oriented strategy and the user will be informed of this change. The new strategy supplies the following partial order of symptoms:

(IN36, OUT24, IN32, OUT24, Valve5Y1, Valve5Y2, Wires)

The values of this symptoms will be asked to the user with regard to the above order (symptoms that are collected in a symptom group, will be asked together). In this case the only diagnosis can be FailureClamping/ReleaseDevice, because there are no further refinements of the context specified.

5.2.2. Refinement Phase

Which more precise relations do exist ? - In this phase an introduction of more detailed diagnoses in the form of intermediate and final diagnoses is carried out.

context: FailureValveOpen	
precondition:	(Valve5Y1 = close)
symptoms:	(Relay5K1, Wires, Diode, Stress)
refinements:	(FailureRelay,FailureWires, FailureDiode,FailureVoltage)
strategy:	default

context: FailureClamping/ReleaseDevice	
refinements:	FailureValveOpen
strategy:	default

symptom: Relay5K1	
type:	(open, close)
possible tests:	Relay-Check

test: Relay-Check	
precondition:	qualification 2
output:	Please check the relay < > !
difficulty:	qualification 2 -> easy to acquire qualification 3 -> very easy to acquire

symptom: Wires	
type:	(ok, broken)
possible tests:	Wire-Resistance-Check
importance:	important

test: Wire-Resistance-Check	
precondition:	qualification 2
output:	Please check the wires < > !
difficulty:	qualification 2 -> less difficult to acquire qualification 3 -> very easy to acquire

symptom: Diode	
type:	(short-circuited, not short-circuited)
possible tests:	Resistance-Check
importance:	very important

test: Resistance-Check	
precondition:	qualification 2
output:	Please check the diode < > !
difficulty:	qualification 2 -> less difficult to acquire qualification 3 -> easy to acquire

symptom: stress	
type:	(existing, not existing)
possible tests:	Voltage-Check

test: Voltage-Check	
precondition:	qualification 2
output:	Please check the voltage < > !
difficulty:	qualification 2 -> less difficult to acquire qualification 3 ->easy to acquire

context: WireBroken	
precondition:	(wires = broken)
refinements:	./.
correction:	Wire-Repair

A system run in this phase may possibly lead to a final diagnosis (and to a proposal for a correction) on a lower level of abstraction, because there are more contexts defined in the slot 'refinement' of the context FailureValveOpen.

5.2.3. Procedural Phase

How to use the objects defined in the previous phases? - The procedural phase introduces strategies and ordering rules for establishing a symptom order.

context: FailureClamping/ReleaseDevice		
Establishing the syptom order with ordering rules:		
(IN38 = logical 1)	-->	(check OUT7)
(OUT7 = logical 0)	-->	(check Valve5Y1)
(Valve5Y1 = close)	-->	(check Valve5Y2)
(Valve5Y2 = open)	-->	(check Wires)
(Wires = ok)	-->	(check IN32)
strategy:	importance-oriented before difficulty-oriented	

MachineFailure strategy <-- importance-oriented, FailureClambing/ReleaseDevice strategy <-- rules
In this phase the system holds more information with regard to the strategies. The uppermost context will now be started with the importance-oriented strategy. The context 'FailureClambing-/ReleaseDevice' will be provided with ordering rules, because the user has decided to go on in accordance to the machine cycles. After each ascertainment of symptoms, the system checks whether the precondition of a context is fulfilled. If this is the case, a context change takes place.
The context 'FailureValveOpen' contains as strategy a combination of the importance and the difficulty-oriented strategy. The connection operator *before* has the following meaning: First determine an order of the symptoms that has details about importance, then apply the difficulty-oriented strategy to the remaining symptoms. It is conceivable that there are more connection operators, but they are not explained here.

5.2.4. Phase of Realization

Which correlations between symptom values do exist? - In this phase the input of constraints for symptom values follows.

context: FailureClamping/ReleaseDevice		
Input of determination rules:		
(Valve5Y1 = closed)	-->	(OUT7 = logical 0) (IN38 = logical 1)
(OUT7 = logical 0)	-->	(IN 38 = logical 1)
strategy:	derivation-oriented before rules	

This context gives now priority to the derivation-oriented strategy. This strategy prefers those symptoms which enable the system to derive other symptom values. At first, according to the connection, Valve5Y1 is asked, because of the state of this valve the values of OUT7 and IN38 can be derived. Otherwise, in the next step, the value of OUT7 will be asked. After this the ordering rules will be evaluated.

6. Towards more Automatical Knowledge Acquisition Methods

This chapter is related to an algorithm for generating determinations [Russell86] described in [Althoff, Kockskämper, Maurer, Stadler, Weß89]. We begin by defining the notion of determinations in the sense of MOLTKE. After giving a extensively modified version of the original algorithm we introduce a knowledge-based extension which enables the method to improve the capabilities of the underlying expert system. Some ideas for the type of the used knowledge are discussed subsequently. Finally, we describe our approach to extend the method to a more powerful one.

6.1. The Definition of Determinations

In MOLTKE determinations represent the relevance of a set of symptoms S1 to a set of symptoms S2, i.e. S1 determines S2 (S1 $\}$ S2), if the fulfillment of S1 allows the derivation of the values of S2. The simple formal definition given by [Russel86] is:

$$P \} Q :\Leftrightarrow \forall z\,[P(z) \wedge Q(z) \rightarrow \forall u\,[P(u) \rightarrow Q(u)]]$$

where, in our sense, the predicate P denotes all those symptoms which are common for some cases and Q some which are not. In the example above P(x) holds if the case x contains symptom S1 and Q(x) holds if case x contains symptom S2. We distinguish two kinds of determinations. A determination is called total if the values of S2 can be inferred from S1 with the probability 1, it is called partial if the probability of the inference is less than 1. The probability is computed by a determination factor δ [Russell86] :

$$\delta = \frac{|\{ x \mid P(x) \wedge Q(x) \}|}{|\{ x \mid P(x) \}|}$$

where P and Q are defined as denoted above. A partition of the interval [0,1] renders the attachment of each determination to one of the following probability categories: total, of-high-probability, probable, of-tolerable-probability, of-low-probability and unprobable, respectively.

6.2. The Syntactic Approach

The approach described in [Althoff, Kockskämper, Maurer, Stadler, Weß89] works on the similarity of concrete empirical cases, i.e. comparing cases with cases. This fact implies a number of problems, especially the main purpose of learning determinations is not taken into account.
Learning determinations will help to improve the actually known best way to do a diagnosis. In this sense the best way is the one which ascertains as few symptoms as possible. Instead, the original method generates as many as possible determinations, hoping that some of them are useful. We now give an improved method which compares cases with diagnoses.
First, we introduce some helpful terms:

(1) A case c is given by a set s of ascertained symptoms and an ascertained

hypothesis h written as $c_h(sm)$[1].

(2) A diagnosis d is given by a set s of symptoms to ascertain and a hypothesis h written as $d_h(sm)$.

(3) Let $d_h(sm_1) \ldots d_h(sm_n)$, $n \geq 1$ be the already known diagnoses of hypothesis h. Then we define the set S_h of all known symptoms for h as $S_h := sm_1 \cup \ldots \cup sm_n$.

(4) The set D_h of possible determinations for hypothesis h basing on S_h is defined as

$$D_h := \{(\{s_1, \ldots, s_n\} \} s) \mid (s, s_1, \ldots, s_n \in S_h) \wedge (s \notin \{s_1, \ldots, s_n\} \wedge (\forall i,j \in [1..n]: (i \neq j) \Rightarrow (s_i \neq s_j)) \}$$

(5) Then, the set D_h of all known or, so to say, learned Determinations for hypothesis h is explained by $D_h \subseteq D_h$

According to our aim of comparing cases with diagnoses, we have to define an order, which works on cases as well as on diagnoses. The subset relation induces a partial order on the set of all cases and diagnoses belonging to one hypothesis.

Definition: $a(sm_1) <_c b(sm_2) :\Leftrightarrow sm_1 \subset sm_2$ with sm_1, sm_2 defined as above and a,b $\in \{c_h, d_h\}$

Now we need to define in which case a diagnosis is minimal according to a given set of symptoms and known determinations basing on the defined order $<_c$:

Definition: A minimal diagnosis $[d_h(sm)]^m$ is defined as follows: Assume S_h, D_h to be defined as above. Let $d_h(sm)$ be some diagnosis. Then $d_h(sm)$ is called minimal if the following condition holds: $sm = S_h - \{\sigma \mid \exists (\{s_1, \ldots, s_n\} \} \sigma) \in D_h \}$

According to the partial order $<_c$, the diagnosis $[d_h(sm)]^m$ is a minimal element.

Now, we give the modified algorithm, which compares the empirical cases to the minimal diagnoses:

Input: A set of cases c_{all} with some hypothesis h_1 to h_n

Output: A Modification of the sets D_{h_i} for all $i \in [1 .. n]$

Algorithm:

Create a partition such that $c_{all} = c_1 \cup \ldots \cup c_n$ with $c_i = \{ c_{h_i}(sm_{i_k}) \mid$ case with hypothesis $h_i \}$.

For all $c_i \in c_{all}$ do

  For all $c_{h_i}(sm_{i_k}) \in c_i$ do

    If $(c_{h_i}(sm_{i_k}) <_c [d_{h_i}(sm)]^m)$ then $D_{h_i} := D_{h_i} \cup (sm_{i_k} \} (sm - sm_{i_k}))$ endif

  endfor

endfor

The determination factor δ is computed as follows

$$\delta = \frac{|\{\chi_{h_i}(\sigma) \mid ([d_{h_i}(sm)]^m <=_c \chi_{h_i}(\sigma)) \wedge (\chi_{h_i}(\sigma) \in c_i)\}| + 1}{|\{\chi_{h_i}(\sigma) \mid (c_{h_i}(sm_{i_k}) <=_c \chi_{h_i}(\sigma)) \wedge (\chi_{h_i}(\sigma) \in c_{all})\}| + 1}$$

It might seem to be a problem to get the initial minimal diagnosis $[d_{h_i}(sm)]^m$, but all the information needed is represented in the described context heterarchy (see chapter 5.).

6.3. Cases and Some Knowledge-based Modifications

To improve the method still further, we will have a closer look to the empirical cases and their types. According to the purpose of the method we can distinguish the following types of cases:

(1) $c_h(sm_1), [d_h(sm_2)]^m$: $sm_1 = sm_2$

---

[1]) We denote a single symptom by 's' and a set of symptoms by 'sm'

(2) $c_h(sm_1), [d_h(sm_2)]^m$: $sm_1 \subset sm_2$

(3) $c_h(sm_1), [d_h(sm_2)]^m$: $sm_1 = (sm_2 \cup \Delta)$ where $\Delta \neq \emptyset$ is a set of additional symptoms; $\Delta \cap sm_2 = \emptyset$

(4) $c_h(sm_1), [d_h(sm_2)]^m$: $sm_1 \subset (sm_2 \cup \Delta)$ where Δ as in (3) and $sm_1 \cap \Delta \neq \emptyset$ holds

Types (1) and (3) can be neglected because they do not represent any improvement of $[d_h(sm_2)]^m$ and therefore there is no possibility to generate any determination. The already described algorithm of 6.2 handles type (2) in a satisfactory manner. The fourth type was not handled yet at all because the syntactical methods are not powerful enough. An examination shows that there are two interpretations to take into account:

(a) The symptoms of $(sm_2 - sm_1)$ are meaningfully replaced by the symptoms of Δ. To motivate this thought imagine, e.g. a service technician who uses a totally different strategy than the expert system. He could try to ascertain symptoms the expert system never would. But nevertheless the technician does a better diagnosis. In this case it is desirable to generate a determination which expresses this capability.

(b) There are only some additional redundant symptoms which have no meaning to the resulting diagnosis.

It is only possible to solve this situation if the system is equipped with some additional knowledge. To handle the appearance of such additional symptoms we follow the approach of introducing the notion of Qualitative Engineering. That means knowledge, in our terminology, on the level of the machine's principal technical organization. For example an information like "a relay is switched by an I/O-state".

Thereby we are able to handle interpretation (a) as follows: If it is possible to find an explanation for the additional symptoms according to the qualitative technical knowledge, then it is justifiable to assume that the actual case has interpretation (a). So the determination $sm_1 \} (sm_2 - sm_1)$ will be generated where the condition $\Delta \subset sm_1$ holds. In principle the symptoms of Δ could be totally new. So it is possible to cover some modifications on the machine's construction, except those which result in actually unknown faults.

On the other hand, if we cannot find an explanation for a symptom of Δ it is not possible to decide whether we are able to handle the actual case as if it has interpretation (b) or not. Because we are not interested in generating an overhead of useless determinations, the operation we apply at the momentary state of our work is to do nothing.

6.4. Extending the Method

6.4.1. Motivation

So far, as described above, the method is strongly restricted. The similarity between two cases is only based on the syntactical equality of symptoms and the use of some explanations. Therefore it is not possible to consider such properties of cases like a more common symmetry. But symmetry can be a very interesting kind of similarity, which leads towards an intensified use of the notions of Qualitative Engineering and determination. This property of symmetry is often found in the currently examined domain of technical diagnosis. A reasonable fact, if the high complexity of this domain is taken into account. Imagine a CNC-machine which has to change the tools automatically. This mechanism has to fulfill two functions among others:

1. clamp a tool
2. release a tool

These two processes are reverse to each other. A tool is released by a hydraulic pressure onto a piston which opens the chuck. Then reducing the pressure clambs the tool. This contrary behavior continues throughout the whole mechanism. But in both situations the same error may occur, for example a piping of the hydraulics breaks or there is a defect in the control system:

Example: A similar case-context pair with reverse symptoms

a context	case no. 59
error-code(i59)	error-code(i34)
switch-position(valve_5Y1,1)	switch-position(valve_5Y1,0)
switch-position(I/O-state_IN36,1)	switch-position(I/O-state_IN36,0)
switch-position(I/O-state_OUT7,1)	switch-position(I/O-state_OUT7,0)
switch-position(valve_5Y2,0)	switch-position(valve_5Y2,1)
state(piping-system,ok)	state(piping-system,ok)
state(chuck,clean)	state(chuck,clean)
switch-position(I/O-state_IN32,1)	switch-position(I/O-state_IN32,0)
defect(I/O-card)	defect(I/O-card)

Now imagine a situation where, e.g. only the last three symptoms and the error-code of case no. 59 are given. Using the algorithm of the preceeding chapters, the comparison of such a case with the given context would not come up with a usable result.

6.4.2. Extension

The unsatisfactory result in the example is caused by the fact that the more complex similarity between the context and the case is not covered. Now, we introduce an extension which does so. The introduced algorithm for syntactical similarity-oriented learning of determinations uses the first definition given below. The similarity is determined by equality only. Our extension introduces an additional general transformation which replaces the simple equality.

Definition (old): Simple Similarity of symptoms. Let **S** be the set of all possible symptoms. The Simple Similarity $=_s$ of two symptoms $s_1, s_2 \in \mathbf{S}$ is defined as follows:

$s_1 =_s s_2 :\Leftrightarrow s_1 = s_2$

Definition (new): Similarity of symptoms. Let **S** be the set of all possible symptoms. The Similarity $=_{si}$ of two symptoms $s_1, s_2 \in \mathbf{S}$ is defined as follows:

$s_1 =_{si} s_2 :\Leftrightarrow \exists t,t' \in \mathbf{T} : [t(s_1) = t'(s_2)]$ where **T** is the set of all introduced transformations with $t \in \mathbf{T} \Rightarrow t : \mathbf{S} \rightarrow \mathbf{S}$

It is easily seen, that the old definition is a specialization of the new one if the identity is chosen as the transformation. Additionally, the definition of the subset relation has to be updated by the substitution of simple equality with our new definition. Now situations like the one mentioned above are covered:

Example: Generating determinations by using transformations

context	case no. 59*
error-code(i59)	error-code(i34)
switch-position(valve_5Y1,1)	switch-position(valve_5Y1,0)
switch-position(I/O-state_IN36,1)	
switch-position(I/O-state_OUT7,1)	
switch-position(valve_5Y2,0)	switch-position(valve_5Y2,1)
state(piping-system,ok)	state(piping-system,ok)
state(chuck,clean)	state(chuck,clean)
switch-position(I/O-state_IN32,1)	switch-position(I/O-state_IN32,0)
defect(I/O-card)	defect(I/O-card)

$$t := \begin{cases} \text{not} & \text{s = switch-position(a,b) with a,b out of the domain of definition of switch-position and not(switch-position(a,b)) := switch-position(a,}\neg\text{b)} \\ \text{id} & \text{else} \end{cases}$$

Using the defined transformation t the algorithm of 6.2. is able to detect the similarity between the context and the case. Then the following symptoms are determined:

t(switch-position(I/O-state_IN36,1)) = switch-position(I/O-state_IN36,0)
t(switch-position(I/O-state_OUT7,1)) = switch-position(I/O-state_OUT7,0)

6.4.3. A Closer Look to Transformations and Qualitative Engineering

The question which now arises is what transformations really are? To give an answer on this we first describe where the transformations come from.
Transformations are gained out of the knowledge we denote as Qualitative Engineering. Remember the example above. There the used "not" can be derived from a fundamental rule which describes a common principle of machine organization: "If the task of a functional part of a machine is to undo an action, then this is normally done by applying the according steps backwards". Adding some basic information about the relations of valves, relais, I/O-state and so on as introduced in 6.3. we are able to generate the not-transformation. (Technically, the necessary knowledge aquisition is done by analyzing the concrete machine cycles using some methods of inductive learning). Thus our notion of transformation represents an operationalization of Qualitative Engineering.
In our sense Qualitative Engineering means to have knowledge about the correlations of basic machine components or, more precise, knowledge about the basic organization of the machine. E.g. valves can be controlled by relays. To use such a kind of knowledge it is not important to know how this control is physically done. Without organizational knowledge the use of transformations would not be practicable.
Qualitative Engineering could also mean to have knowledge about basic machine components and their fundamental function. E.g. a valve can be represented as a component whose function is to regulate material flows. To use such a kind of knowledge it is not important to consider properties of a special instance of a valve. At the moment we have not integrated this interpretation in our methods.

6.4.4. Usability

To show that this extension is useful, let us mention some of its advantages and their reasons:
Size - Less cases are needed to get a "good" set of determinations. In the example from 6.4.2 using the old, simpler definition of similarity causes the need of an additional case which fits better to the context than the given one. A comparison to case 59 would not supply the correct symptoms. The consequence is, that applying the extended definition causes a smaller case base to do good determination learning.
Competence - In addition to the preceding part (6.3), which described the use of the notion of Qualitative Engineering for the generation of explanations as an extension of the underlying syntactical method, the method´s competence increases by introducing transformations. As shown in the example, the use of the new definition implies applying additional knowledge not only to avoid making errors but also to increase the method´s power.
Necessity - It is possible to take advantage of similarities by introducing new meta-rules into the expert system which infer the rules of diagnosis for contexts like the given one, from the rules of a similar context by applying transformations. A smaller knowledge base will be the result. Keeping this in mind, it is necessary to have the ability of recognizing cases which are similar to a context in the sense of the introduced meta-knowledge. Otherwise the knowledge would get filled up with redundant and useless determinations.
Efficiency - The efficiency of the method increases because operationalising the additional knowledge implies not to have to generate an often used explanation for each symptom or case which is affected.

7. State of Realization

The KA mechanism has been realized to meet the main requirements our engineer had stated during the development of the knowledge base [Kockskämper89]. The component for the generation of determinations is currently under development [Traphöner89], [Wernicke89]. We will start soon with the implementation of the extension of this method and the integration of Qualitative Engineering into this component. There already exists a case-based reasoning system in MOLTKE [Althoff, Kockskämper, Maurer, Stadler, Weß89], [Stadler, Weß89] which is used in addition to the basic diagnosis system partially introduced in chapter 5. MOLTKE is implemented in Smalltalk-80 and runs on all workstations and personal computers on which the corresponding virtual machine is available [Althoff, Faupel, Nökel, Rehbold89].

8. Conclusion

The KA tool we presented in chapter 5 is intended to serve as a supporting mechanism for further works. Based on a successful integration of the methods for processing experience, introduced in chapter 6 and the underlying diagnosis system, we are actually working on, we hope to get towards a more automated knowledge acquisition. This includes at first, the automatical adaptation of the diagnosis system to a modified CNC machining center, and, at second, the vague possibility to generate a technical diagnosis system automatically by using a detailed machine model in connection with our case-based learning approach enabled by Qualitative Engineering. (See also [Rehbold89])

9. Acknowledgements

We would like to thank Michael M. Richter for introducing the idea of using Qualitative Engineering for our purposes in the above mentioned manner. Finally we would like to thank Klaus Nökel and Robert Rehbold for many constructive discussions within the MOLTKE project, which were essential for the initiation of some new approaches.

10. References

[Althoff, Faupel, Nökel, Rehbold89]
Althoff, K.-D., Faupel, B., Nökel, K., Rehbold, R.: MOLTKE.- to appear in: Extended abstracts of the workshop on knowledge-based diagnosis systems, GMD, Bonn, April 1989

[Althoff, Kockskämper, Maurer, Stadler, Weß89]
Althoff, K.-D., Kockskämper, S., Maurer, F., Stadler, M., Weß, S.: Ein System zur fallbasierten Wissensverarbeitung in technischen Diagnosesituationen.- Proceedings ÖGAI-Jahrestagung, 1989

[Kockskämper89]
Kockskämper,S.: Diskussion möglicher Wissensrepräsentations- und Inferenzmechanismen zur Fehlerdiagnose eines CNC-Bearbeitungszentrums und deren Implementierung als Expertensystem in Smalltalk-80.- diploma thesis, University of Kaiserslautern , 1989 (forthcoming)

[Pfeifer, Held, Faupel88]
Pfeifer, T., Held, H.-J., Faupel, B.: Aufbau einer Wissensbasis für Fehlerdiagnosesysteme von Bearbeitungszentren. VDI-Z VDI-Verlag 10/88.

[Rehbold89] Rehbold, R.: Deriving Causal Rules from Structure Descriptions in a Technical Diagnosis Domain, SEKI-Report 89-1 University of Kaiserslautern, 1989

[Russel86] Russel, S.J.: Analogical and Inductive Reasoning.- Ph.D.-Thesis, Stanford University, Dez.1986

[Stadler, Weß89]
Stadler, M., Weß, S.: Konzept und Implementierung eines fallbasierten, analogieorientierten Inferenzmechanismus und dessen Integration in ein regelbasiertes Expertensystem zur Diagnose eines CNC-Bearbeitungszentrums.- project thesis, University of Kaiserslautern, 1989

[Traphöner89] Traphöner, R.: Ein Konzept für die Verarbeitung von Erfahrungswissen in MOLTKE.- project thesis, University of Kaiserslautern, 1989 (forthcoming)

[Wernicke89] Wernicke, W.: Ein System zur Verarbeitung von Erfahrungswissen in MOLTKE.- diploma thesis, University of Kaiserslautern, 1989 (forthcoming)

On 'Extracting knowledge from text': Modelling the architecture of language users

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

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

Abstract

We propose a unified view of natural language understanding and knowledge acquisition. Knowledge is not 'extracted' from a text, but rather is *added* to the text by a 'cogniting' agent. The text, and whatever is contained in it, serves only as a triggering mechanism. This process of addition is concept cluster attachment. This can be generalized to artificial notations such as mathematical formulas and diagrams (even Rorschach tests!) and general signing such as facial expressions and gestures. We develop a minimal three-level architecture for a cogniting agent, consisting of verbal, conceptual, and sub-conceptual levels.

We further propose that natural language understanding and knowledge acquisition from text requires expertise. We discuss how this expertise may be acquired and incorporated into an expert system, and incrementally build up the architecture of the theoretical version of such an expert system, which we call LUKES. We discuss its implementation as LOGOS using the sortal analysis tool SORTAL.

1 Introduction

> *What is controversial is the contention that a concept is something mental and proprietary to a particular mind: for if I am to understand what you say I must attach the same meaning to your words as you do.*
>
> *—Anthony Quinton explaining conceptualism [Bullock et al 1988, p. 160]*

Intuitively, one feels that there should be a close connection between natural language understanding (NLU) as studied in computational linguistics, and knowledge acquisition (KA) as studied in the context of knowledge-based systems. Yet the connection between NLU and KA is seldom described explicitly. Even natural language–mediated KA [Regoczei and Plantinga 1987, Regoczei and Hirst 1989] dodges the issue.

In this paper, we would like to tackle this problem by developing requirements specifications for a system that performs both natural language understanding and knowledge acquisition from text. Our strategy is to incrementally build an architecture for a theoretical (*gedanken*) system that we will call LUKES (for Language Understanding and Knowledge Extraction System). The enlarging of the specification for LUKES is guided by the question of what features and components such a system would have to have to do a good job.

Thus we will examine the difficulties associated with NLU and KA. As we come to an understanding of what makes a particular subtask difficult, we posit a module or an operationally or computationally effective strategy to cope with the difficulty. This gives the final version of LUKES a very comprehensive architecture. While some components of this architecture may not be implementable at the present date for reasons of theoretical or technological limitations, or resource limitations, LUKES nevertheless serves as a benchmark against which the NLU component of KA tools and expert system tools can be compared. Also, if at some future date a toolmaker wants to include an NLU component in their KA tool, LUKES provides a comprehensive shopping list of 'must-have', 'should-have', and 'nice-to-have' features.

We take the implementation issue seriously, including the problem of building up large knowledge bases in order to acquire the knowledge for other, even larger knowledge bases. This bootstrapping operation is addressed through the combination of LOGOS and SORTAL in section 7.2. The implementability of LOGOS, as a KA system with NLU capabilities, is discussed.

The building up of the architecture for LUKES through a series of versions of incrementally greater sophistication and power is accomplished through a series of steps. Each increment is motivated by a theoretical discussion of some severe difficulty in NLU, KA, cognitive modelling, or conceptual analysis. A solution to the difficulty is postulated, and the computationally or

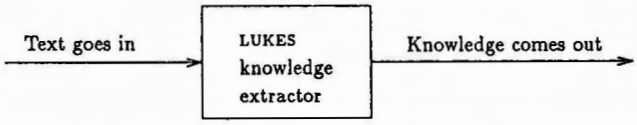


Figure 1: Knowledge acquisition made easy.

constructively operationalized solution is incorporated into the architecture of LUKES as a feature or a separate module. In this way we hope to build up a well-motivated, realistic, rich architecture that addresses the real problems, not merely the toy, laboratory issues that ignore the difficulties of practice.

To help the reader focus on the essential point of the paper, we should mention at the outset that the central mechanism that we will put forward both for NLU and KA is **concept cluster attachment** (CCA) [Regoczei and Hirst 1988], which we see as unifying NLU and KA. In addition, CCA generalizes from text understanding to the understanding of a broad range of symbols, signs, and other means of communication. We hope that this helps to throw some light on a severe difficulty in KA, namely the oft-repeated objection that the KA analyst does not only work with what the informant says, but with extra-linguistic clues such as gestures and facial expressions.

Throughout the paper we emphasize the role of **the agent**, and in particular, we incorporate into LUKES a model of the cognitive architecture of the agent. The agents that are being modelled are not only the one that is doing the NLU and the KA but also those that are mentioned during the NLU or KA process.

Because of space limitations, the present paper can do no more than outline certain key components of LUKES and refer briefly to the rest, in the hope that the overall argument will still be cogent. This paper is one of a series on **domain of discourse creation, conceptual analysis, the meaning triangle, multiple domains of discourse,** and the uses of **sortal analysis** and SORTAL [Regoczei and Plantinga 1987, Regoczei and Hirst 1988, Regoczei and Hirst 1989, Regoczei and Hirst in prep].

2 Knowledge acquisition as mining

2.1 Taking a metaphor literally

Phrases such as *extracting knowledge from text* are metaphors. Such metaphors are very useful, but may be misleading in certain crucial circumstances. The longer phrase, *extracting and modelling knowledge from text*, taken from the announcement of this conference, raises an additional issue: is it possible to 'model from text'? Can modelling be 'from' something? Is there an **operationalizable** interpretation of these phrases? That is, can we write an algorithm for the kind of KA activity that these phrases refer to that would run on an algorithmic machine?

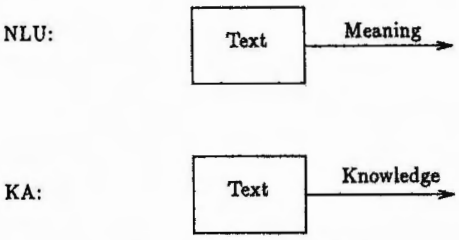


Figure 2: NLU parallels KA.

The metaphor *knowledge extraction from text* suggests a sort of mining operation. We dig into the text, cut out the little bits of knowledge, and throw away the debris. Perhaps the 'ore' that results needs further refining before being cast into ingots of pure knowledge.

Now, some people are more expert at mining for knowledge—*i.e.*, text understanding—than others, and this varies with context, domain knowledge, and medium of communication. Therefore, according to the basic tenets of knowledge-based systems research, these people's expertise should be acquirable, and should be able to be built into an expert system. Let us take the metaphor literally. Figure 1 shows LUKES, our *gedanken* system for the task: text goes in and knowledge comes out.

There is a parallel between natural language understanding and knowledge acquisition from text. In the case of KA, we start off with text (be it printed text or an interview with an expert) and 'extract knowledge from it'. In NLU, we also start with text and 'find meaning in it' (figure 2). There seems to be a close similarity, so let us propose a strategy: let us identify 'meaning' with 'knowledge', thereby creating a parallel

between the two activities. The support for this strategy is to be found in the ease with which we will use insights from one of the fields to help in the analysis of the other.

2.2 Operationalizing metaphors

In the computational context, **operationalizability** of a task refers to the feasibility of writing algorithms that will run on an algorithmic machine to perform the task. Thus operationalizability entails being able to find effective procedures. Operationalizability is closely linked to constructivism in mathematics. For example, in Euclidean geometry, trisection of an angle is a perfectly well understood notion, but it is not operationalizable because it cannot be performed with a compass and straightedge. In arithmetic, construction of *all* the natural numbers is not operationalizable, though constructing the first few, or even the first few million, is.

In physics, Bridgman [1950] based the ontology of physical entities on experiment and measurement. A physical quantity exists if effective measures can be taken and physical operations can be performed that monitor the effect of the hypothesized physical entities. In general, as described by Rapaport [1953], operationalism links thought with action and insists on specification of actions to be carried out as giving meaning to both linguistic utterances and theoretical entities.

It is in this spirit that we are seeking, but at present only partially finding, an operational interpretation of NLU and KA from text.

3 A three-level architecture

3.1 Three levels of cogniting

To establish our terminology, we start with a simplified model of 'cogniting' agents. Our neologism *cogniting* is a verbal form of *cognition* [Regoczei and Plantinga 1987]. Both computers and people are agents, but (so far) only people are cogniting agents. We would like to get computers to do more cogniting. (In contrast, bricks are not agents, nor do they cognit.)

A cogniting agent can be seen as operating at three levels: verbal, conceptual, and sub-conceptual (see figure 3). The **verbal** level includes the syntactic component of language use, essentially handling language as character strings without any regard for meaning, significance, or the concepts 'behind' the words. The **conceptual** level involves the construction of conceptual mental models, and the manipulation of these models to understand the external world.

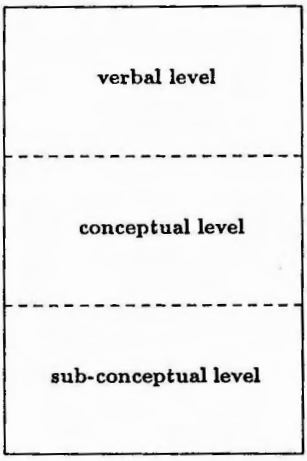


Figure 3: Three levels of cogniting.

The **sub-conceptual** level of the agent includes its physiological embodiment and functioning, as well as all the emotions, gut feelings, and vague hunches that are very much part of human cognitive activity.

We can picture verbal activity as being at the surface level, with the deeper level of meaning provided by either the conceptual or the sub-conceptual level. At the conceptual level, we can build conceptual models of sub-conceptual affects,[1] external-world entities, and the meaning 'behind' the words.

The main purpose of the three-level architecture is to separate the conceptual from the verbal, and to distinguish both of these from the rest, *i.e.*, from the sub-conceptual. If LUKES is to be a cogniting agent, it too must have such an architecture.

3.2 A minimal architecture for a cogniting agent

On the basis of the discussion above, we now propose a model for a language-using cogniting agent that in a sense is a minimal architecture: it contains components (though not necessarily all of them) that must be present to give an account of what happens in language use, language understanding, knowledge acquisition, and cognition in general.

Our model is shown in figure 4. The model receives both verbal and non-verbal input from the world, and can respond verbally or non-verbally.

[1] Do not misread *affects* as *effects*.

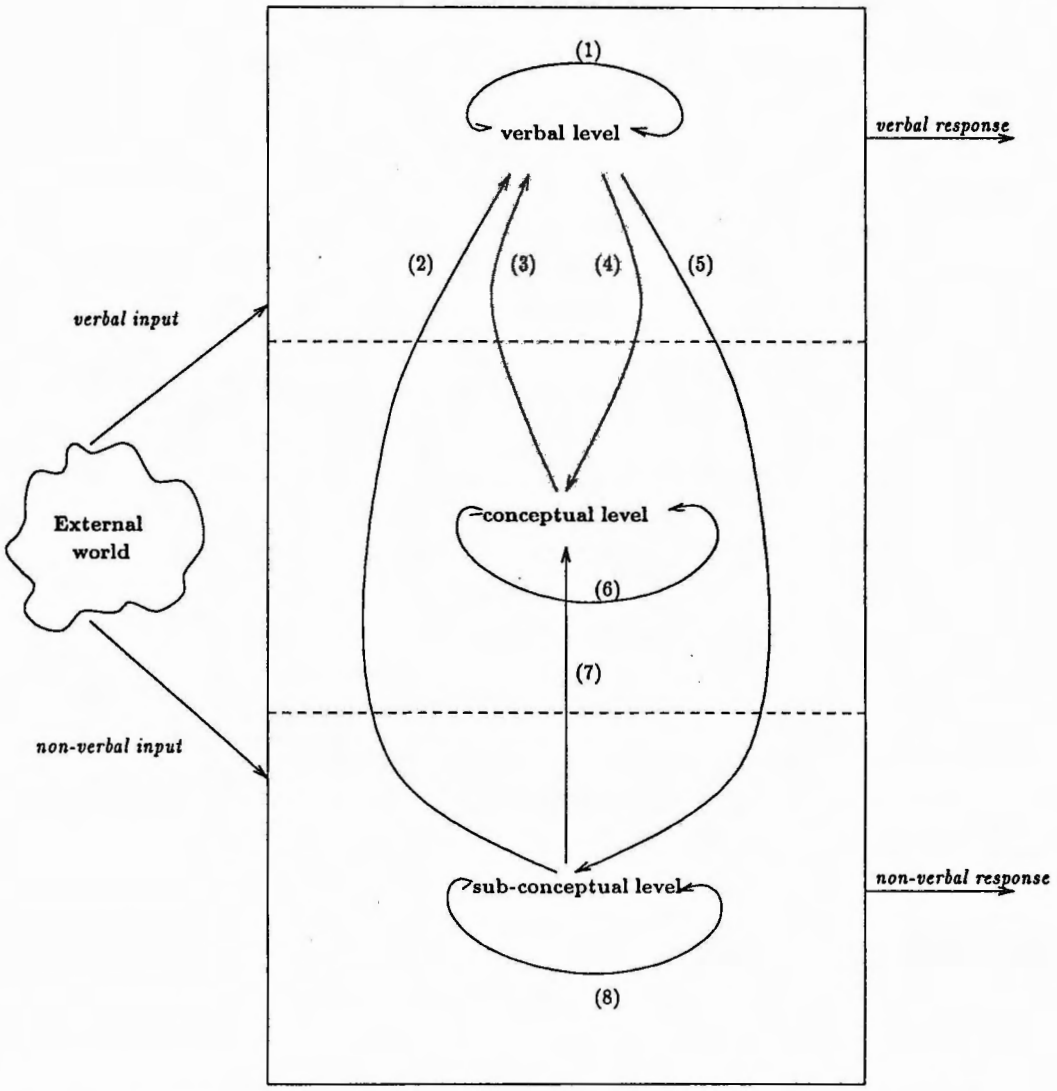


Figure 4: Minimal architecture for a cogniting agent. The numbers are explained in section 3.2.

The non-verbal input could be visual or tactile, for example, and the non-verbal output could be physical actions. The numbered arrows show operations within and between levels, as follows:

(1) A verbal feedback loop, as in talking to oneself.

(2) Verbal input producing a sub-conceptual affect.

(3) Attaching concepts to words.

(4) Using conceptual structures and models to generate text.

(5) Verbalizing a sub-conceptual affect (*e.g.*, a cry of pain).

(6) A conceptual feedback loop: thinking in concepts, not words.

(7) Building a conceptual model of sub-conceptual affects.

(8) A sub-conceptual feedback loop: sub-conceptual thought (*e.g.*, making aesthetic judgements).

This is a simple, but adequate model. It may seem complicated, but it is the smallest model that suffices: three levels, one external entity, and eight operations. The only extension that might be convenient to have is a pre-processor that distinguishes verbal from non-verbal input (figure 5). This helps us to talk about the sorting, *e.g.*, how do we know that a piece of visual stimulus is actually text.[2]

To motivate our model, we mention a simple but important example from the field of communications. Much advertising, whether verbal or pictorial in nature, is directed to the sub-conceptual level of the reader. This sub-conceptual affect is often very hard to describe—it takes a good deal of analysis to be able to make the emotional reaction clear. In fact, what happens is that a conceptual model is built that reflects the sub-conceptual affect. In articulation at the verbal level, it is this conceptual model that gets described. Likewise, verbal input can act as a stimulus for the formation of conceptual structures and mental models of a conceptual kind. These in turn can be described verbally. Thus the conceptual level is central both in mediating between the sub-conceptual and the external world and supplying the deep-structure-level meaning for text. Concept cluster attachment, to be discussed in section 4.6, provides the mechanism of interaction between the surface, verbal level and the deep-structure, conceptual level.

[2] We are not claiming any psychological reality for such a pre-processor. On the contrary, humans almost certainly process all input in parallel as both verbal and non-verbal. For example, in the well-known Stroop colour-naming task [Stroop 1935], subjects are unable to fail to read the words they are shown, even though their task is only to name the colour of the ink in which they are written.

3.3 Implications for LUKES

Following this architecture, LUKES would have to have three levels (figure 6): a verbal level that is essentially free of non-linguistic knowledge, a conceptual level using, say, conceptual graphs [Sowa 1984], and a sub-conceptual level that uses either a connectionist model or spreading activation in a network. Because LUKES is a computer program, all input is text—strings of characters—going to the verbal level. However, some strings are treated as natural language prose and some as (names and descriptions of) concepts. As output, LUKES would produce knowledge bases, with the knowledge encoded as (symbolic representations of) conceptual structures.

4 Comparing different theories of knowledge

In this section, we compare some different theories of knowledge/meaning and text understanding. We elaborate on these to motivate our own theory of understanding as **concept cluster attachment** (CCA). CCA is not a rival theory to any of the others; it can coexist, with greater or lesser harmony, with each of them. Rather, it is a synthesis, for all the other theories point to CCA as an essential component of their operationalizability. We will try to show that concept cluster attachment is the main mechanism in, and the common mechanism between, natural language understanding, artificial notation understanding, general sign understanding, and knowledge acquisition.

4.1 The content theory of knowledge

The **content theory of knowledge** is 'intuitively obvious'. The theory holds that some texts are more meaningful, more informative, more capable of conveying knowledge, because they have more **content**. This content, often referred to as the **meaning**, is what is 'extracted' from the text by the knowledgeable reader. The knowledge is there in the text, ready to be seen, extracted, and acquired. This is a coherent and perfectly satisfactory position to take to explain how meaning is conveyed from author to reader. It only breaks down when we try to implement it on a

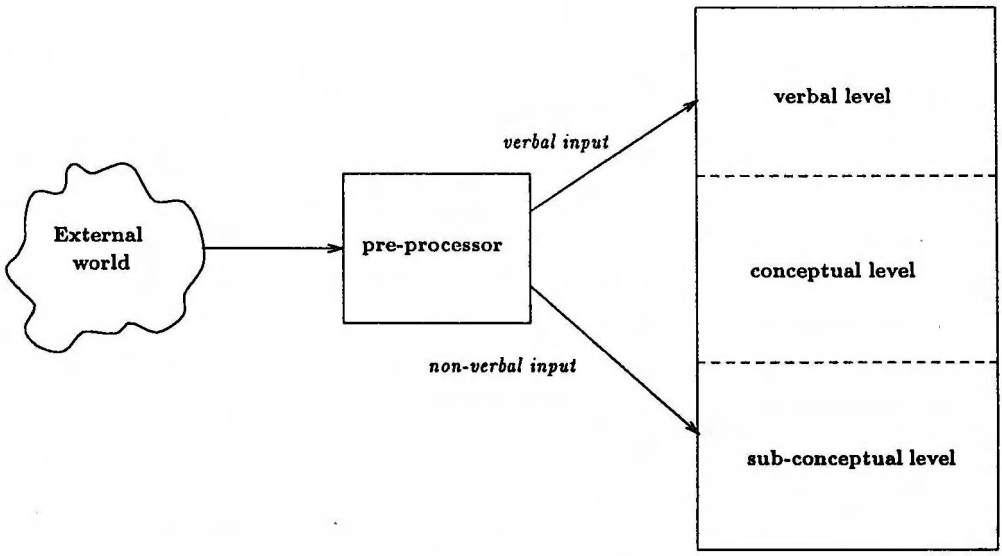


Figure 5: Adding a pre-processor that distinguishes verbal from non-verbal input.

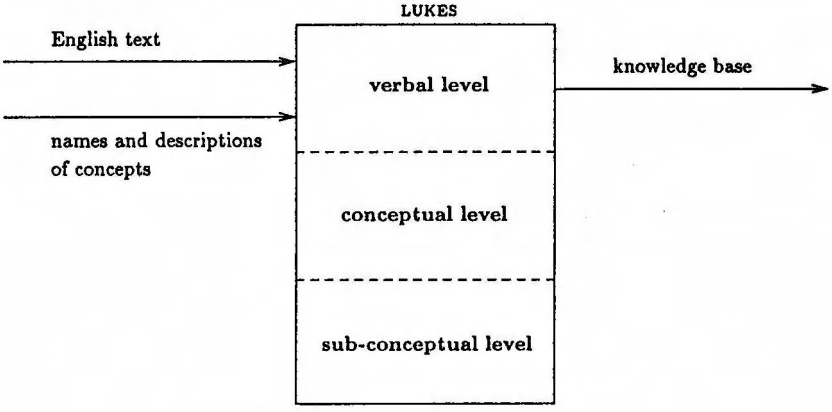


Figure 6: A three-level architecture for LUKES.

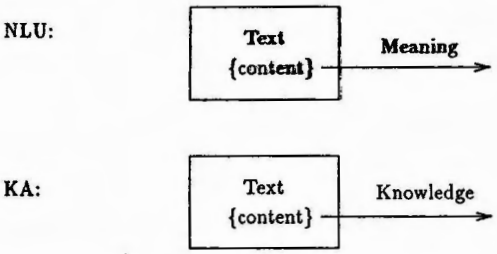


Figure 7: The content theory of knowledge.

computer, or we are confronted by an individual who sees things differently, and does not find quite the same 'obvious' content in the text that we find and surmise that the author intended. In other words, the content theory works just fine until it breaks down. *Extracting knowledge from text* is a phrase firmly founded in the content theory of meaning. It is one of the 'metaphors we live by'.

In what follows, we accept the appropriateness of the content theory of knowledge or meaning, but want to extend it or reinterpret it for cases where it is not operationalizable or it breaks down. What exactly is wrong with the content theory? Mainly, it cannot be operationalized in a computationally effective way. Whatever the 'content' of the text is, it is not pure, refined knowledge or meaning. Rather, the content is best construed as a set of clues as to what kind of association can be made with the text. We are not enquiring about one single individual on one particular occasion; we are asking about the normally expectable associations. This requires knowledge. We have to know what knowledge can be attached—where the word *can* is to be construed as *allowable within a domain of discourse.*

To summarize: Text is seen as possessing a content, and the knowledge or meaning derived from text (see figure 7) is nothing more than the extraction of this content.

4.2 The representation theory of knowledge

The **representation theory of knowledge** holds that natural language text does not so much contain knowledge as **represent** it—almost like a pointer, it points to it. In other words, the significance of texts, notations, and signs comes from what they **stand for**. For example, a fine car represents the wealth of its owner. The car, as a general sign, is something that semioticians and cultural and communication specialists would study. The understanding of such a sign, or the knowledge to be acquired from the sign, or the 'content' of the sign, stems from the representational role that it plays.

Like the content theory of knowledge, this is a coherent position. To make it operationalizable, we must first realize that the car does not 'represent' all by itself; there must be an interpreter, a cogniting agent. The car acts as a **clue** or **trigger** that activates certain processes within the agent. We can hypothesize some simple models for these processes. For example we might say that the interpreter has a rule in her head:

> **if** car is large and expensive-looking
> **then** owner is rich.

This is part of the knowledge base of the interpreter, a result of socio-cultural training and the background of the agent. In the absence of an interpreter, the car is not representing anything.

4.3 The hermeneutical theory of knowledge

In the **hermeneutical** tradition, knowledge of the true meaning of text is the 'holy grail' of the **hermeneutical process**. This is just a technical term for a way of thinking that has a strong grip on the popular imagination. The core of the theory is that to understand text, one performs a search for its 'true meaning'. As with the content theory, the meaning is 'in' there—but getting it out is not easy. The popular analogy is to a sculpture that is 'in' a block of granite, and all that the sculptor has to do is get it out of there. This is a powerful image, and a very potent metaphor, easily applicable to text understanding. But it is a misleading image. There is another kind of sculpture: an assemblage of metal, produced with a blowtorch, both adding and subtracting pieces to or from the original material. If understanding is like sculpting, it is like this second kind of sculpting.

We mention these issues because of the powerful hold that the search for the 'true meaning of the text' had for a long time on computational linguistics. This manifested itself by the desire to be able to *deduce* the meaning of a text from the character string alone. This paradigm is fading, to be replaced by various knowledge-based processes.

The hermeneutical theory is the strongest challenge to concept cluster attachment. Its proponents would say that if (as we concede in sections 4.6–4.7) you are allowed to attach to a text any old concept cluster you like, you can go right ahead and do so—but that won't make it the one correct

true meaning. The contest thus is concept cluster attachment versus 'the true meaning of the text'.

4.4 The information-theoretic theory of knowledge

Just as we identified knowledge with meaning, we can identify both knowledge and meaning with **information** in the Shannon–Weaver [1949] sense. On this view, information is a substance, like energy, that can be 'sent'. In fact, it can be measured, which may be the strongest evidence of its 'true existence'. The **information-theoretic** approach supports, and is supported by, powerful images and metaphors centred on communication as transmitting a message from a sender to a receiver. What these metaphors conceal is the detailed operationalized mechanism that Shannon offered [1948]: selecting from a pre-defined set of alternatives. That is, the receiver is 'pre-wired' to act or respond in accordance with the pre-defined messages that can be sent. The messages could include a request that the agent alter or 'rewire' itself—as, for example, in an ASCII transmission in which the ESC character signals the receiver to shift to a different set of actions. But there are many different kinds of alterations one might want the agent to make—to rewire itself, say, to store uninterpreted strings, or to accept a description of a new set of responses, or even to change the way in which it accepts such descriptions. The theory is not amenable to this. It would be like trying to reprogram a pocket calculator by entering numbers.

Thus we see the difference between a signalling theory such as Shannon's and a semantic/epistemological theory that requires not only the ability of the agent to rewire itself but also a knowledge base in the agent to make possible a higher-level interpretation of the signal. Shannon is quite explicit [1948, p. 3] that he is not concerned with meaning *per se*, but rather is focusing on the physical channel that conveys information that may, in turn 'package' a kind of meaning. His analysis is thus at a different level from the one at which concept cluster attachment takes place. For example, one can correctly perceive a text in, say, German, character by character, but that's not the same as understanding it.

4.5 Miscellaneous theories

There are many other theories of knowledge that we cannot give space to here. These include the **mental-models theory** (which we used in [Regoczei and Hirst 1989]), the **deduceability theory** alluded to in section 4.3, and a **surrogate theory** that may also be called 'representational'. In the surrogate theory, the text is a surrogate for the knowledge in it. When we look at a text, the theory says, we aren't looking at knowledge, but we are as good as doing so.

We mention these theories for the sake of completeness; the objections that they raise against our position can be countered much as we did with the theories above.

4.6 The attachment theory of knowledge

We turn now to the approach that we will use in LUKES, a synthesis of the theories we have reviewed.

The question *Is there knowledge in the text?* can be answered in three ways:

1. Yes, there is knowledge in the text. This is the content theory or the hermeneutical theory.
2. No, there is no knowledge in the text. Text is lexical, knowledge is conceptual. To attribute conceptual entities as somehow 'contained' in lexical entities is a category mistake. This position cannot explain why some texts seem more 'meaningful' and better suited for the purposes of communicating than others.
3. A compromise position would claim that there is *something* in the text, even if that something cannot be characterized as knowledge. Perhaps this something is information, or clues that the astute reader, who is an expert in language understanding, picks up and uses in the process of understanding.

One does not have to take a position on these three views, because a synthesis is possible. We note that both 1 and 3 require special expertise: the first, the expertise of being able to see the content in the text, and the third, the expertise of being able to **attach** concepts to text, given the clues contained in it. Thus there is no knowledge *per se* 'in' text; what there is is a set of perceivable clues that stimulate the reader to synthesize knowledge, attach it to the text, and even attribute it to the author. This is what we are calling **concept cluster attachment.**

Variant positions can be developed. For example, one could argue that a certain amount of knowledge *is* perceived as being 'in' the text, and to this meagre stock of initial knowledge a greater amount of knowledge is added. This would be another form of the concept cluster attachment process.

Notation	*Meaning*
cat	word
[CAT]	concept
⟪[CAT], [DOG], ...⟫	concept cluster
⟪[CAT] | [DOG], ...⟫	concept cluster centred on the concept [CAT]

Figure 8: Notation used in concept cluster attachment.

What exactly are these clues in the text? They need be nothing more than substrings. It is quite remarkable that when presented with text as a character string, such as *The cat sat on the mat*, the substring *cat* will be associated with a concept such as [FELINE], whereas most readers of this paper will look at the character string *Felugrott a macska az asztalra* and associate nothing more with the substring *macska* than [NONSENSE], [GIBBERISH], or [HUNGARIAN]. The clues may also be construed as concepts themselves, *i.e.*, the sparse initial knowledge that might be seen in the text.

CCA enables us to enlarge our set of techniques for linguistic ambiguity resolution [Hirst 1987, Hirst 1988]. For example, consider the sentence [Hirst 1987, p. 112] *Ross was escorted from the bar to the dock*. Each of the underlined words is ambiguous and has more than one reading that is consistent with a reading of the other. In the older paradigm of the deduction theory of knowledge, there was no way of deducing a unique interpretation from the text alone. Now that we can attach concepts to text, we know that the ordered pairs

⟨*Ross was escorted from the bar to the dock*, [COURTROOM]⟩

and

⟨*Ross was escorted from the bar to the dock*, [HARBOURSIDE]⟩

are of considerable help. Assisting an NLU program by feeding it concepts would have been considered cheating in the earlier paradigm, which restricted processing strictly to the verbal level. Given a central concept, such as [COURTROOM] or [HARBOURSIDE], an entire concept cluster centred on the concept, can be attached to the sentence. It doesn't matter whether the concept clues are supplied externally or perceived as part of the initial knowledge in the text; in either case CCA starts with a small amount of knowledge and generates a larger amount.

The arbitrary nature of CCA is immediately evident, and has to be addressed. Could wildly different things be added to the sentence? Yes, and they often are. These spurious additions are not weeded out until the interlocutors have harmonized their mental models. Often they are not weeded out at all, but are killed and thrown away along with the rest of the microdomain that was being formed as part of the harmonization process.

We need a theory like CCA to explain how an otherwise meaningful text can be given to somebody who fails to understand it or fails to find the meaning 'contained' in the text. The phenomenon of 'not-understanding' is of great significance. In CCA theory, not-understanding is equivalent to not being able to attach any concepts to the text. For example, when the reader is confronted by the words *kutya, farkas, róka*, she may baffled—no concepts 'come to mind'. This contrasts sharply with what does happen when the reader is confronted with *dog, wolf, fox*. Any coherent theory of NLU or KA has to come to grips with the question of what does or doesn't happen when we *don't* understand something.

In figure 8, we summarize our notation for words, concepts, and concept clusters.

In what follows, we focus on the position that the knowledge, or at least the right level of knowledge, is not in the text, but has to be added by an agent. We then explore further the mechanism of concept cluster attachment, and the expertise required to be good at it.

4.7 Examples of concept cluster attachment

The terms *concept*, *concept cluster*, *concept cluster centred on a concept*, and *attachment of concept clusters to text* require some clarification. A concept is a cognitive construct [Regoczei and Hirst 1989], a component of a mental model used to understand the world. These concepts are specific to an agent. They are typically personal versions of public concepts [Regoczei and Hirst 1988]. A concept cluster is a loose collection or set of concepts. We use this as an informal precursor to the more formal and structured mechanisms such as conceptual graphs [Sowa 1984]. As for the other terms, the best way to indicate what they mean is through examples.

A good way to illustrate concept cluster attachment and its fundamentally arbitrary nature is

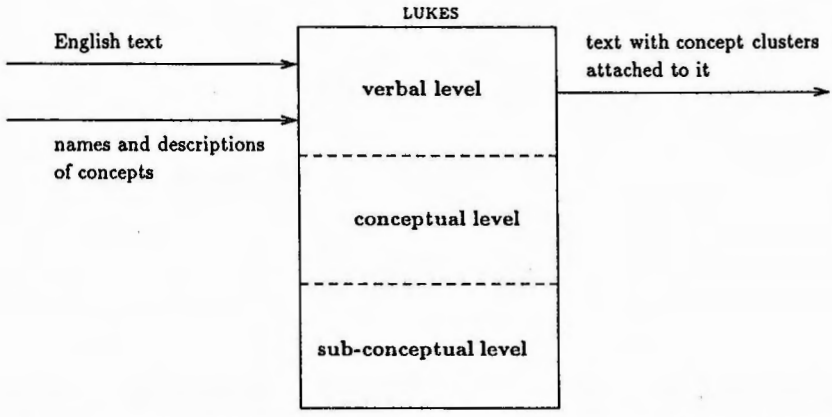


Figure 9: LUKES with concept cluster attachment. The output consists of ordered pairs, $\langle T, C \rangle$.

through the old Rorschach-test joke: The new recruit goes to the army psychiatrist who shows him a Rorschach inkblot and asks, "What does this remind you of?" "Sex," answers the recruit. All of the other inkblots get the same response. The punchline of the joke varies in different versions;[3] here, we only need note that the recruit effected a concept cluster attachment, expressed in our notation as $\langle inkblot_i, [\text{SEX}] \rangle$. It is futile to ask whether sex is in the inkblot (content theory), whether it symbolizes sex (representation theory), whether sex is the true meaning of the inkblot (hermeneutical theory), or whether there is an information component in the inkblot message that can be decoded as sex (information-theoretic theory). No, the attachment has more to do with the pre-occupation of the young recruit than anything else. In spite of this, however, we may accept the claim that in some sense at least, the recruit's concept cluster attachment was completely arbitrary.

Cutting down on the arbitrariness of the attachment is a social process—in particular, a **harmonization of mental models** [Regoczei and Hirst 1989]. The uninitiated would attach no concept to the word *kutya*, but those expert at handling such words would attach a concept cluster such as

$$\langle\langle\langle [\text{DOG}] \mid [\text{PET}], \ldots \rangle\rangle\rangle$$

(if Hungarian) or

$$\langle\langle\langle [\text{PORRIDGE}] \mid [\text{CHRISTMAS}], \ldots \rangle\rangle\rangle$$

(if Ukrainian).

There is a social process of training that draws it into an individual's head that the word *cat*, for example, is to be associated with a cluster such as

$$\langle\langle\langle [\text{TABBY}], [\text{TOMCAT}], [\text{PERSIAN}], \ldots \rangle\rangle\rangle$$

but not

$$\langle\langle\langle [\text{PETAL}], [\text{SEPAL}], [\text{PISTIL}], \ldots \rangle\rangle\rangle$$

The knowledge base that enables one to effect the concept cluster attachments that are 'normal', 'accepted', 'common', or 'according to the dictionary' may look like the conceptual lexicon given by Sowa [1984]. Acquiring such a knowledge base is a long learning experience. Endowing an NLU system with such a knowledge base is a basic necessity.

4.8 Implications for LUKES

Concept cluster attachment suggests a new approach to knowledge bases. A knowledge base is now a collection of concept clusters attached to text. So knowledge is captured both at the verbal level and the conceptual level. The architecture of LUKES is now modified as follows: The output is sortally composite entities consisting of ordered pairs $\langle T, C \rangle$ where the first component, T, is a lexical entity and the second, C, is a set of concept clusters (figure 9).

4.9 Unifying natural language, artificial notation, and signs of a general nature with concept cluster attachment

By natural language we mean not just ordinary text, as it is usually found in 'natural' con-

[3] For example: Psychiatrist: "I think you have a fixation on sex." Recruit: "*I* have a fixation on sex?! *You're* the one who's been showing me all these dirty pictures!"

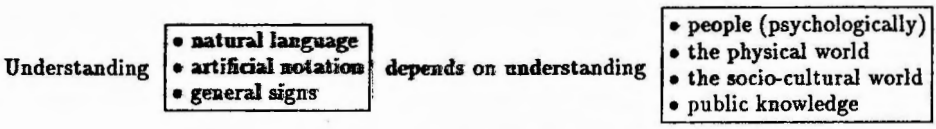


Figure 10: Two kinds of understanding.

texts, but also technical sublanguages such as those of medicine, law, politics, and mathematics. **Artificial notations** are also often used within the context of a sublanguage, but may transcend sublanguages. They are often 'language-like', but need not be; they can be like two-dimensional diagrams, relying upon spatial arrangement on a page for effect. Examples include calculus notation, first-order logic, dataflow diagrams, blueprints, and conceptual graphs. Such notations are artificial, *i.e.*, deliberately invented, unlike natural language.

By signs of a general nature we mean any non-verbal communication. This includes human actions such as gestures, facial expression, or other 'body language' (sic!). But it also includes things like the communicative aspects of clothing, three-dimensional models of airplanes, and punching someone in the mouth.

All of these we want to treat in a unified way. While they are in some ways different, they are amenable to a uniform treatment as they all have a suggestive power that makes them usable for communication. However, our comments about artificial notation and diagrams or general signs are subsidiary points to illustrate the universality of our approach. Mostly, we restrict our attention to natural language text, and the knowledge or meaning that may be 'extracted'.

We explicitly posit concept cluster attachment as an explanatory mechanism for the understanding all these kinds of communication in response to the problem that the KA analyst does not only work with what the informant says, but also picks up extra-linguistic clues from the behaviour, intonation, and gestures of the informant. It has been an irritant of long standing in KA that while the importance of these extra-linguistic clues was recognized, and they can be even captured by tape recorder, or, better still, by video recorder, how they were to be 'translated' into knowledge to be represented in a knowledge base was totally obscure.

5 The pessimistic part

NLU or KA from text is an inherently difficult process. The various theories above are not final solutions to the problem but little more than indications of the multi-faceted approach that has to be taken. Our suggestion of emphasis on the attachment theory of knowledge, in particular, concept cluster attachment and attribution, is a synthesis—a promising research direction and a way of developing software architectures. In putting it forward, we are not blind to the severe difficulties of NLU and KA from text. In this section, we discuss these difficulties, and in section 6 suggest a way out that is based on CCA.

5.1 Understanding understanding: A crucial distinction

There is a crucial difference between understanding sentences and understanding people. Most often, we understand each other without the necessity of using language. In fact, the use of language indicates that understanding broke down [Regoczei 1987]. Often the difficulty of understanding a sentence boils down to knowing what is 'behind' the sentence, who the agent that uttered it is, what fragment of the physical or socio-cultural world it is about, and what bodies of public knowledge are being referenced (see figure 10).

These problems can be placed in a general context that illuminates both the problem of 'finding meaning in the text' and the 'extraction of knowledge from text'. Understanding the items in the left column of figure 10 often depends on understanding items in the right column. The physical world or the socio-cultural world can be understood through direct observation, or being informed through private versions of the public knowledge. This distinction captures the 'aboutness' of signs. They refer to something outside themselves. Understanding the signs (left column) has to be preceded by understanding what the signs are about (right column). The difficulty of understanding text entails the even greater difficulty of understanding the world.

5.2 Understanding text

Quite apart from the difficulties discussed in the previous section, there are severe difficulties just at the text level:

Syntax: Natural language is syntactically complex, and syntactic structure and semantic interpretation each depend upon the other. In addition, much natural text is syntactically ill-formed.

Ambiguity: Words, syntactic structures, and modes of reference can all be ambiguous. The listener has to decide among the possibilities.

Semantic problems: Utterances can be vague, *i.e.*, with no fully determined meaning, or involve non-literal devices such as metaphor and metonymy.

Reference: Pronouns and other definite references must be resolved.

Discourse connections: The meaning of a paragraph is more than just that of its sentences, but also involves the relationship between them. The relationship is usually not explicit, but must be determined by the listener. Likewise, a longer text is more than just the sum of its paragraphs.

Research in computational linguistics is addressing all these issues, and there are already many interesting (partial) solutions.

6 The optimistic part

The expert-system approach assumes that a difficult activity can be handled by a system with a knowledge base that incorporates expertise. Then the difficulty of performing the task reduces to the difficulty of building the knowledge base. We will follow this paradigm by investigating how to turn LUKES into an expert system.

Given LUKES's three-level architecture, it needs a knowledge base at each level. Furthermore, it needs a knowledge base for each of the major activities that require special expertise, such as the operations of CCA, formulation of mental models for sub-conceptual affects, and expression of conceptual models in verbal form. Following the expert-system paradigm, the best way to build up the knowledge base is by asking an expert to tell us how she does it. This we do in the next section.

6.1 Interviewing the expert

Most people are experts at understanding their native tongue, yet when asked to describe how they understand text, they become surprisingly tongue-tied.

For example, we may go to an expert/informant with the well-known example *The cat sat on the mat* and ask her how she goes about understanding it or determining what it means.

Expert: I don't know. It doesn't mean anything. I can't tell you what it means unless you tell me what you are talking about. What are you talking about?

Analyst: Well, actually I didn't mean anything by it. It was just an example that I used.

Expert: Well, if it's empty words, and you don't mean anything by it, then how should I be able to determine the meaning?

Analyst: Well, what would people usually mean by it? How would you expect other people to understand this phrase?

Expert: I wouldn't. I recognize *The cat sat on the mat* as a silly example that philosophers use when they don't want to face issues about how people really use language in their everyday life.

In other words, the expert applied a great deal of knowledge to contextualize and determine that the string is a fake, just empty words. There is no meaning in it at all.

With a different expert, we might get a very different response.

Expert: *The cat sat on the mat*? There's not much here to understand, but I recognize the word *cat*. It looks like English. I do know quite a bit about cats. I've owned a few cats, and I know what a cat is but I don't know if they're talking about a specific cat in this case. My cats didn't sit on the mat. They usually preferred an armchair or the newspaper that I wanted to read. I remember the time that ...

The analysis of this seemingly simple dialogue is quite revealing of the strategies people use to make sense of text. For example, the text string is repeated to make sure the listener heard it correctly. A remark is made at the lexical level about the word *cat* being English. Next, the referent level is broached in claiming possession of world knowledge about cats. Specific instances are cited (personal ownership) and the phrase claiming to "know what a cat is" is a claim about having possession of the concept [CAT]. Finally, the expert draws a distinction between understanding the sentence as opposed to understanding the

speaker, *i.e.*, she does not know what the speaker means but indicates that she would like to continue talking by mentioning some particular feline behaviour concerning armchairs and newspapers.

Thus, careful analysis reveals a surprising richness in all this. The expert/informant does concept cluster attachment, sortal analysis, conceptual analysis, conceptual modelling, contextualization, categorization, and modelling the agents involved. She is hypothesising agents as needed (see section 6.2). The processes performed by the expert/informant can be modelled, interpreted, or pictured by drawing diagrams (*e.g.*, agent-centred meaning triangles—see [Regoczei and Hirst 1988]). We model the agent's modelling of agents and their mental models.

6.2 Positing other agents

There is a special form of expertise that helps us, experts at NLU, to cope with the difficulty of understanding not only sentences but each other—the ability to imagine what the other person is thinking and, so to speak, direct our communication to this hypothetical agent. In particular, and this is perhaps the most significant activity, one hypothesizes into being prototypical public agents, attributing to them 'normal expectations'. In other words, using the terminology of mental models, not only do we form mental models to understand the world, but we also form mental models of other agents' mental models. It is this ability that enables us to apply Gricean rules [Grice 1975], use conversational implicature, or change style to conform to the (posited) domain of discourse that others are using.

LUKES should have the facility to hypothesize cogniting agents, with the three-level-architecture specifying and distinguishing:

- what the posited agent *says*;
- what conceptual structures the posited agent may have associated with the words he or she uses; and
- what sub-conceptual states the agent may be in.

This ability of LUKES would carry out knowledge acquisition as attribution, and perform version control of the changing state of the posited agent as the text is read or the dialogue progresses.

The arbitrariness of CCA, discussed above, is overcome by the facility of being able to harmonize mental models; more precisely, the mental models as posited by LUKES are attributed to the agent and attached to the text received from the agent.

6.3 LUKES as an expert system

The concept cluster attachment module of LUKES is the most important component. To generate knowledge bases, LUKES needs a wide range of knowledge on how to attach concept clusters. Looking upon it as an expert system, CCA is its main expertise—that is, its expertise is in knowledge base formation from textual input.

Given the three-level architecture for a cogniting agent that we described in section 3, we can now list the main components of LUKES:

1. A verbal module.
2. A conceptual module.
3. A sub-conceptual module.
4. A concept cluster attachment module, consisting of:
   (a) an episodic microdomain creation submodule; and
   (b) a domain of discourse creation submodule.
5. A module to form conceptual models of sub-conceptual states.
6. A text generation module to output a symbolic description of a knowledge base.
7. A social interaction module, consisting of:
   (a) an agent-positing submodule; and
   (b) an inventory of posited agent surrogates.

In the next section, we address the implementability of such a system.

7 Implementation

7.1 What constitutes implementability?

The implementability of a piece of software that exists only in the form of a wish-list, *i.e.*, a requirements specification and perhaps an architecture, has to be investigated using certain stringent, practical criteria. We can consider such software as implementable (but not yet implemented) if:

1. The architecture is modularized;
2. Each module can be prototyped;
3. There is a mechanism for scaling up from the prototype to a production version; and
4. A reasonable estimate of time and money requirements for the scaling up is obtainable.

While we tried to provide a rich enough architecture for LUKES, and even looked at some prototyping possibilities inspired by the Absity system

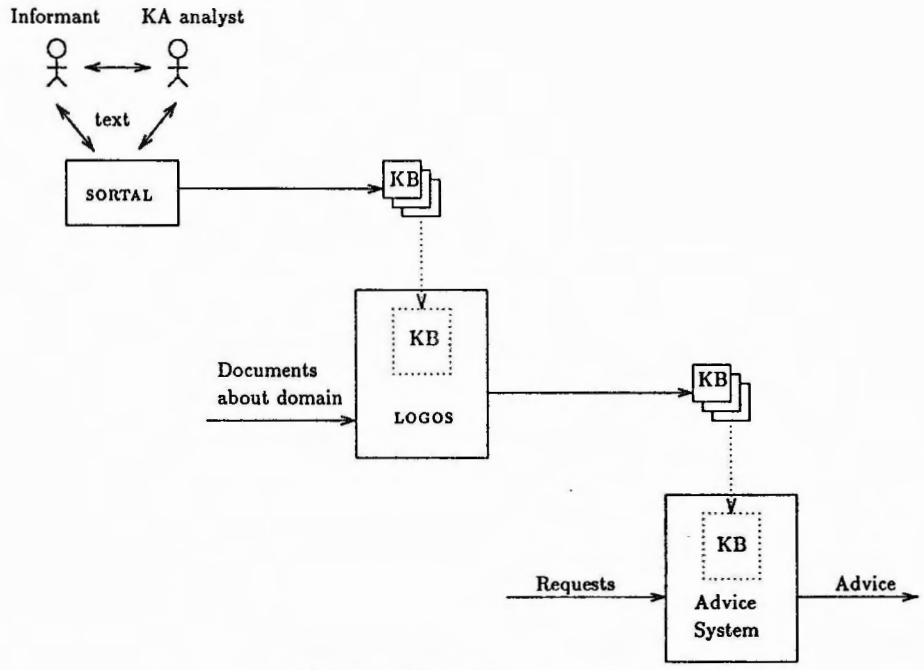


Figure 11: Bootstrapping LOGOS.

[Hirst 1987, Hirst 1988], there are serious obstacles in the way of achieving task 3 above, and hence task 4 is not possible. In the rest of this section, we investigate a direction towards implementability and some of the difficulties.

7.2 Implementing LOGOS supported by SORTAL

We will now briefly outline a possibly-implementable version of LUKES, which we will call LOGOS ("In the beginning, was the word ... "). LOGOS is a system whose architecture is derived from that of LUKES by cutting LUKES down to an implementable configuration. This necessarily entails the loss of certain capabilities and a departure from the realistic functioning of cognitive agents.

For implementation, we will drop the subconceptual module, because at present there is no clear way of interfacing a connectionist model of computation with an algorithmic one. LOGOS emphasizes the CCA module, with rich capabilities to create domains of discourse, including multiple domains and episodic microdomains, possibly using a concept-cluster version of Hirst's Polaroid Words mechanism [Hirst 1987, Hirst 1988]. The generation of knowledge bases, including knowledge at both the textual and conceptual levels (but expressed as symbols, *i.e.*, character strings) requires a text/symbol–generating module. To read documents, extract knowledge from text, and monitor the informant-analyst–based, natural language–mediated knowledge acquisition process, with version control of the emerging domain of discourse, requires a 'social interaction' module that posits models of language-using agents.

Because these modules are all knowledge-based, building up the knowledge bases for LOGOS becomes an important issue. We propose to use SORTAL [Regoczei and Hirst in prep], a knowledge acquisition tool that assists the analyst in sortal analysis [Regoczei and Hirst 1988], to get LOGOS started on its path of acquiring knowledge (see figure 11). SORTAL is a program to be used by a KA analyst to help set up a knowledge base with a small sample of representative text—priming the pump. The basic concepts, established using SORTAL, are passed to LOGOS. LOGOS reads the domain documents to set up the knowledge base for, say, an advisor system.

8 Conclusion

In this paper, we outlined a three-level architecture for LUKES, a theoretical natural language understanding, knowledge extraction system based upon concept cluster attachment. We discussed how LUKES could be made implementable as LOGOS by cutting its architecture down.

Experts may have hundreds, even thousands, of different kinds of techniques that they use for CCA. Our intention was not to catalogue these, but to point to a new research direction that:

1. Unites NLU and KA;
2. Places concept cluster attachment in the centre of investigations;
3. Makes the expert-system paradigm central, in that it claims that both NLU and KA can be modelled as expert systems;
4. Asks what kind of knowledge has to be in the knowledge bases of the expert system for NLU and KA; and
5. Explores these issues in the context of developing an architecture for LUKES.

In addition, with CCA being at the centre of the understanding process, there is no reason to treat natural language as a special case. Whatever we state about attaching concept clusters to natural language texts, we can say about attaching them to artificial notations such as mathematical symbols, logical formulas, and diagrams, and general signs, such as facial expressions and gestures.

Some concept clusters have privileged status. There is a long tradition of distinguishing meaning from allusion, or denotation from connotation. From the point of view of CCA, there is no difference. From the point of view of evaluation of the output, *i.e.*, the composite entity $\langle T, C \rangle$, there could be a wide range of 'correctness' ratings according to different, well-established valuations.

The theory presented here is intended to be a descriptive model, depicting what people actually do. However, we have to note that it is a formalized model, and hence deviates from the realistic messiness of the world. People are seldom aware that they are attaching concept clusters, and even less frequently do they manage to keep the separate clusters apart. In this respect, our model is too tidy. In actual, experiential terms, people just 'catch on', 'manage to grasp things', and in a bumbling sort of way, figure out 'what stuff means'. Computers, however, have to take a neater route.

Acknowledgements

Preparation of this paper was supported by grants from the Trent University Committee on Research and the Natural Sciences and Engineering Research Council of Canada.

References

[Bridgman 1950] Bridgman, Percy Williams (1950). *Reflections of a physicist.* NY: Philosophical Library.

[Bullock *et al* 1988] Bullock, Alan; Stallybrass, Oliver; and Trombley, Stephen (editors) (1988). *The Fontana dictionary of modern thought* (second edition). London: Fontana Press.

[Grice 1975] Grice, H. Paul (1975). "Logic and conversation." In: Cole, Peter and Morgan, Jerry L. (editors). *Syntax and semantics 3: Speech acts.* NY: Academic Press. 41–58.

[Hirst 1987] Hirst, Graeme (1987). *Semantic interpretation and the resolution of ambiguity* (Studies in natural language understanding). Cambridge University Press.

[Hirst 1988] Hirst, Graeme (1988). "Semantic interpretation and ambiguity." *Artificial intelligence*, 34(2), March 1988, 131–177.

[Rapaport 1953] Rapaport, Anatol (1953). *Operational philosophy: Integrating knowledge and action.* NY: Harper & Brothers.

[Regoczei 1987] Regoczei, Stephen (1987). Review of: Winograd, Terry and Flores, Fernando. *Computers and cognition: A new foundation for design*, Norwood, NJ: Ablex Publishing Company, 1985. In: *Computational linguistics*, 13(3–4), July–December 1987, 340–344.

[Regoczei and Hirst 1988] Regoczei, Stephen and Hirst, Graeme (1988). "The meaning triangle as a tool for the acquisition of abstract, conceptual knowledge". *Proceedings, Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November 1988. To appear in *International journal of man–machine studies*, 1989.

[Regoczei and Hirst 1989] Regoczei, Stephen and Hirst, Graeme (1989). "Knowledge acquisition as knowledge explication by conceptual analysis." In: Sowa, John F; Foo, Norman; and Rao, Anand (editors). *Conceptual graphs for knowledge systems* [to appear]. Also published as technical report CSRI-205, Computer Systems Research Institute, University of Toronto, January 1988.

[Regoczei and Hirst in prep] Regoczei, Stephen and Hirst, Graeme (1989). "Sortal analysis with SORTAL, a software assistant for knowledge acquisition." Technical report, Department of Computer Science, University of Toronto, in preparation.

[Regoczei and Plantinga 1987] Regoczei, Stephen and Plantinga, Edwin Peter Owen (1987). "Creating the domain of discourse: Ontology and inventory." *International journal of man-machine studies*, 27(3), September 1987, 235-250.

[Shannon 1948] Shannon, Claude Elwood (1948). "The mathematical theory of communication." *Bell System technical journal*, ?(?), July and October 1948. Reprinted in [Shannon and Weaver 1949], 3-91.

[Shannon and Weaver 1949] Shannon, Claude Elwood and Weaver, Warren (1949). *The mathematical theory of communication*. Urbana: The University of Illinois Press.

[Sowa 1984] Sowa, John F (1984). *Conceptual structures: Information processing in mind and machine*. Reading, MA: Addison-Wesley (The systems programming series).

[Stroop 1935] Stroop, J.L. (1935). "Studies of interference in serial verbal reactions." *Journal of experimental psychology*, **18**, 1935, 643-662.

KNOWLEDGE ACQUISITION BY CAUSAL MODEL AND META-KNOWLEDGE

CHARLET J., GASCUEL O.

INSERM U.194 et Service d'Informatique Médicale
Département Intelligence Artificielle et Médecine
91, Bd de l'Hôpital
75634 PARIS CEDEX 13
FRANCE

Tel: (33) 1 45-83-67-28
Fax: 45-86-56-85
Email: charlet@frsim51 (bitnet)

Abstract :

The current trend in expert systems is to use powerful knowledge representations. This is a convenient way to have highly structured and concise knowledge bases. It allows rich functional models to be represented at different levels of abstractation. Such models may be referred as "deep knowledge". Even when these models are refined, they are incomplete for achieving diagnostic tasks. Empiric knowledge (shallow knowledge) must be integrated in order to achieve correct performance.

We provide a way for making the deep knowledge complete before the interpretation by the inference engine. We introduce mechanisms which : (i) interpret the deep knowledge; (ii) exhibit ambiguities and incompleteness; and, (iii) ask the expert about them, and so allow shallow knowledge acquisition. These mechanisms build an operative knowledge base.

These mechanisms are implemented in the LÉZARD essential expert system, currently applied in medicine to diagnose acute abdominal pains.

1 Introduction

High-level knowledge representation tools are widely used in expert systems. In particular, semantic networks including hierarchical and causal links are commonly employed. Using these links, different relevant levels of abstraction and complex models can be easily represented (e.g., [Szolovits et al. 88] CASNET [Kulikowski & Weiss 82], ABEL [Patil et al. 82], SESAM [Ferrand 84], ANGINA [Long et al. 84]). This knowledge is a *deep* one and is opposite to the *shallow knowledge* [Chandrasekaran & Mittal 83]. The description of the knowledge base through a *deep* model

makes knowledge acquisition easy. For example, describing a taxonomy is easier than detailing all the underlying links.

However, interpreting *deep knowledge* is difficult. In the same way as the physician knows *heuristics*[1] for interpreting his own knowledge, it is necessary to give to the inference engine *heuristic strategies* which are required for correctly and efficiently exploiting the knowledge base. In diagnostic decision making, most of the systems are more or less based on the two following *strategies* [Szolovits et al. 88]:

- focus on the highest abstraction levels: the program tries to decide if the most general diagnostic classes are present before deciding on more particular classes. This *heuristic strategy* is directly connected to the knowledge representation in terms of taxonomies.
- unicity or simplicity of the fault process: the program makes attempts to create a unique, causal chain which explains the observed manifestations.

The purpose of these two *strategies* is to reduce the search space and to consider only a few diagnoses at the same time. These *heuristic strategies* are tools for interpreting uncertain, incomplete, and complex knowledge.

Whatever this interpretation may be, it cannot cover up the knowledge incompleteness, and more information is required. As an example, let us consider the three situations A, B, and C, where B may be caused by A, and C may be caused by B (a *deep knowledge*). An interpretation program assuming a unique process (the second *strategy*) will tend to produce the causal chain: C is caused by B, and B is caused by A. But in some cases there are incompatibilities between A and C. That may be expressed by the two production rules : "A $\Rightarrow$ not C", and "C $\Rightarrow$ not A". Any interpretation mechanism would need this knowledge. Typically, this knowledge is *shallow knowledge*.

Unlike *deep knowledge*, *shallow knowledge* is unambiguous in that it accepts only one interpretation. Therefor, *shallow knowledge* interpretation may be computationally rapid and safe. However, *shallow knowledge* is difficult to acquire directly from the expert since it has too small granulation and is too extended.

Van de Velde [Van de Velde 86] proposes *shallow knowledge* acquisition by learning from examples. Starting with *deep knowledge* and a set of real world cases, his program builds a collection of production rules which constitute the *shallow knowledge*.

The LÉZARD system uses a different approach which consists in acquiring *shallow knowledge* from the *deep* one by interacting with the expert. The idea is to translate all the *deep knowledge* into *shallow knowledge*. During this translation, ambiguities are systematically removed. As an additional feature, our system paraphrases the expert. That allows the consistency and the correctness of the *deep knowledge* to be verified.

Outline of the paper

Part 2 is devoted to an overview of the LÉZARD expert system. The general architecture of the system and the knowledge representation it uses are given.

Part 3 considers the basic properties of *shallow knowledge* and *deep knowledge*.

[1] "Heuristic" is employed here to represent all empiric knowledge, i.d. *shallow knowledge* and *meta-knowledge*, which leads the knowledge acquisition process, or controls the inference process. This last *meta-knowledge* is also a *strategy*.

Part 4 details the interactive translation mechanisms, from *deep* to *shallow knowledge*, and show why and how this translation is a knowledge acquisition process.

Part 5 discusses various aspects of our approach.

The LÉZARD expert system is currently applied in medicine to diagnose acute abdominal pains. The examples used in this paper for illustrating the purpose are inspired by this application. However, LÉZARD may be applied to various diagnostic tasks such as car fault diagnosis.

2 Overview of the Lézard system

The system architecture is modular. It may be represented as follows (figure 1):

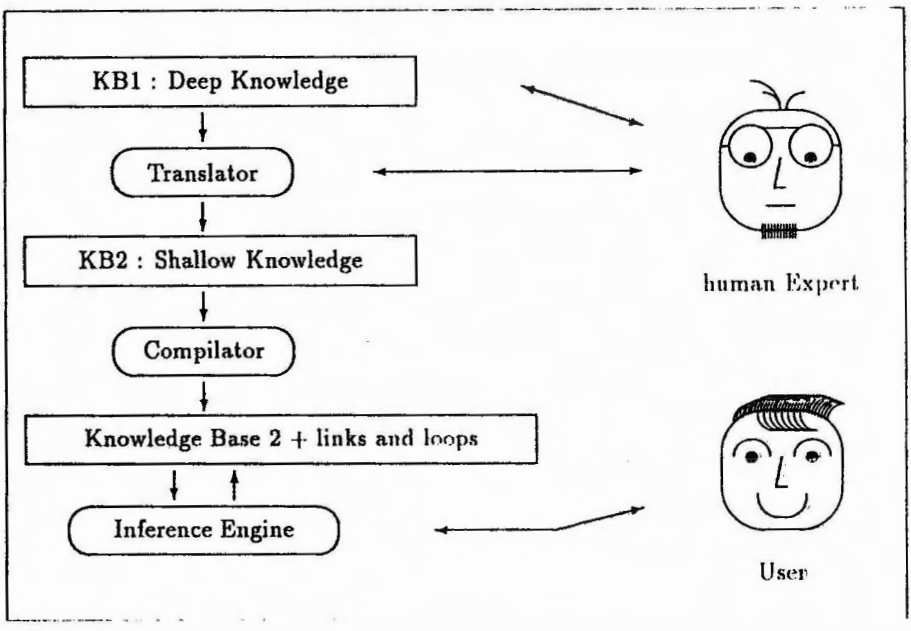


Figure 1: The architecture of the LÉZARD system.

The top level language – Deep knowledge

The *deep knowledge* base is written in a high level language which is as concise as possible. Two kinds of knowledge may be distinguished :

(i) The static knowledge. Its representation is done using a semantic network. This network is made of causal and hierarchical relationships between facts (in medicine facts are signs, syndromes, diagnoses, etc...). The figure 2 shows a small part of the network used on the application to acute abdominal pains. There are two kinds of taxonomies : one by localizaion (in black), the other by process. The causal link : "A :::> B" (represented by a dotted line in the figure) means that A is a possible cause of B. This network is expressed in the LÉZARD formalism as follows:

- Two hierarchical relationships:

  APPENDICITIS and ORGANIC-OBSTRUCTION are-the DIGESTIVE-DISEASE
  FUNCTIONAL-OBSTRUCTION and ORGANIC-OBSTRUCTION are-the OBSTRUCTION

- A causal relationship :

  FUNCTIONAL-OBSTRUCTION is-caused-by PERITONITIS or URINARY-TRACT-INFECTION

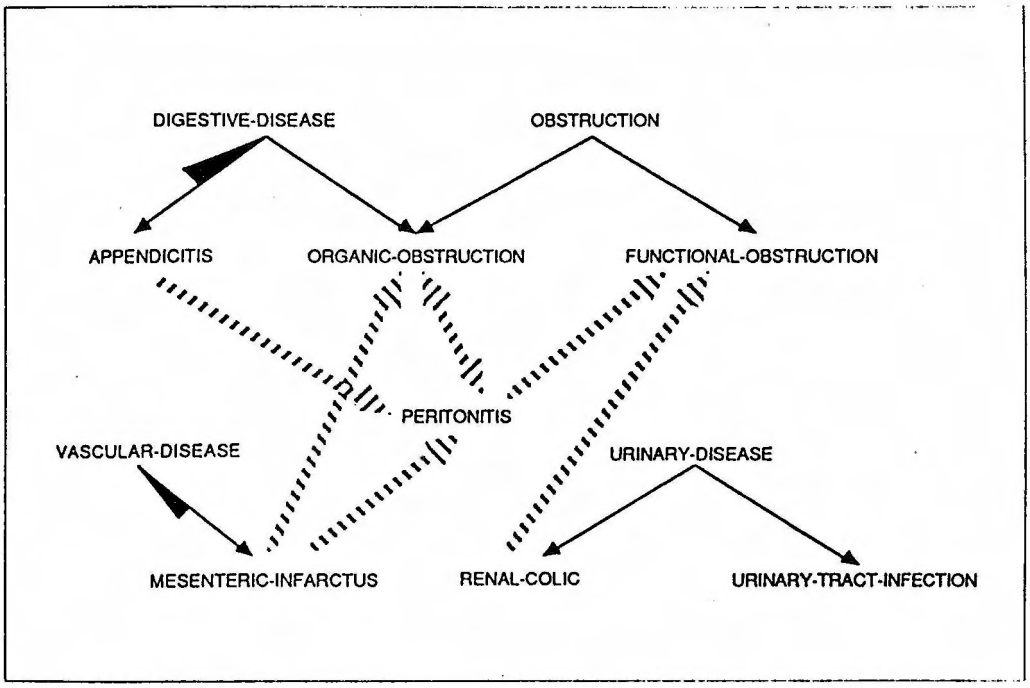


Figure 2: Semantic network representation. Dotted lines represent causal links. Other lines are taxonomic links.

(ii) The dynamic knowledge. It is expressed in a propositional, logic-like formalism. In medicine it is used for representing the relationships between signs and diagnoses; between syndromes and diagnoses; and more generally between facts and diagnoses. A piece of dynamic knowledge is the following:

FEVER evokes APPENDICITIS or PERITONITIS or ...

its meaning is close to (FEVER $\Rightarrow$APPENDICITIS or PERITONITIS or ...). But the implication is not certain (fever is sometimes encountered in other diagnoses).

It must be noted that the dynamic knowledge is not necessarily a *deep* one. It may be *shallow* as well.

The low level language – Shallow knowledge

The TRANSLATOR converts the *deep* model into a *shallow* formalism of the following type: "(and P_1 P_2 ... P_n) $\Rightarrow$ C". This is a production rule where P_i and C are literals in the logical meaning.

The translator

The TRANSLATOR enables the program to map from one level language to another. When mapping, the TRANSLATOR extracts a set of rules from KB1 which constitute KB2. For example, the taxonomy:

FUNCTIONAL-OBSTRUCTION ORGANIC-OBSTRUCTION are-the OBSTRUCTION

is translated into few rules (see below, Part 4.2). Some of these are:

(*not* OBSTRUCTION) $\Rightarrow$ (*not* FUNCTIONAL-OBSTRUCTION)
(*not* OBSTRUCTION) $\Rightarrow$ (*not* ORGANIC-OBSTRUCTION)
FUNCTIONAL-OBSTRUCTION $\Rightarrow$ OBSTRUCTION

The basic idea of the TRANSLATOR is is to transform schematic knowledge of KB1 (*deep knowledge*) into a set of operational knowledge (*shallow knowledge*), which is KB2. During this phase, knowledge is refined interactively with an expert. *Meta-knowledge* leads expert interview.

To give an idea of the work performed by the TRANSLATOR for our application: KB1 contains approximately 135 lines of knowledge and KB2 contains about 1200 rules. This illustrate why expressing complex knowledge, directly as production rules, is illusory.

The compilator

The rules of the *shallow knowledge* base are compiled in order to speed up the interpretation process. In this phase, the program exhibits : the relationships between facts and rules; the partial ordering links between rules; and, the rules' loops. This compilation presents analogies with the compilation of SAM [Gascuel 85] and PROSPECTOR [konolige 79].

The inference engine

The inference engine uses forward chaining and non-monotonic reasoning. It is relatively simple because of the simplicity of the *shallow knowledge* syntax. It has been described in [Gascuel et al. 85].

3 Deep and Shallow Knowledge

While solving a problem human experts often draw on many different kinds of knowledge. What usually makes an expert an expert is *shallow*, *compiled* or *empirical* knowledge . However this kind of knowledge is often just a short-cut, or a compilation of a deeper understanding of the problem domain. This other kind of knowledge is referred to as *deep*, *functional* or *physical* knowledge [Fink 87].

3.1 Shallow Knowledge

Production rules as "A ⇒ non C" express *shallow knowledge*, allowing the expert to build *shallow* models. Their main characteristics are the following [Koton 85; Steels & Van de Velde 85; Fink 87; Klein & Finin 87] :

- their conclusions are drawn directly from observed facts that characterize a situation;
- they directly encode the empirical knowledge that experts use in performing their reasoning task; so they are efficient inference vectors in terms of reasoning rapidity.

However, we have to cope with their following inherent disadvantages:

- they are difficult to extract from experts because of their number and high granularity. Moreover, how they interact is hard to understand; so an eventually existing underlying model is rarely specified;
- the represented knowledge may be difficult to maintain since what is conceptually a single piece of knowledge may be systematically distributed across several objects in a knowledge base;
- by construction, they can give justifications[2] to any reasoning process. As these justifications are based on the physical chain of inferences rather than the meaning of the reasoning, they provide explanations[3] which often are incomprehensible short-cuts;
- they are inflexible, unable to deal with circumstances. In addition, they often contain implicit preconditions. For example, they often require the precondition: "all other things being equal".

Production rules characterize *first generation* expert systems (e.g., MYCIN [Shortliffe 76] or PROSPECTOR [Konolige 79]). Their limitations lead to *second generation* systems whose knowledge base includes empirical rules as their predecessors, but also *deep knowledge*.

3.2 Deep knowledge

Deep knowledge allows representation of causal models. These models describe, with different levels of abstraction the role of each appearing element and represent the causal relationships between their different properties [Steels & Van de Velde 85]. In medicine, causal models are physiological and pathophysiological models whose elements are physiological entities and medical diagnoses [Szolovits et al. 88]. The main characteristics of *deep* models are the following [Koton 85; Klein & Finin 87; Fink 87]:

- the models are deeply explicited;
- they produce justifications as well as satisfactory causal explanations;
- their completeness is easier to verify;

[2] "Justifications" has the meaning usually given in the TMS terminology: the justification of a proposition P express the belief attached to the different propositions appearing in the reasoning leading to P.

[3] The "Explanation" of a proposition expresses the reasoning understanding.

- they exhibit correct performances at the periphery of their domain: There is a graceful performance degradation because the *deep* models cover a larger domain than *shallow knowledge*.
- However, it is generally admitted that they are slower and more complex than *shallow knowledge* since a more sophisticated control and interpretation structure is required.

Defaults and advantages of these two kinds of knowledge explain why *second generation* expert systems have both. *Deep knowledge* is used for two major task:

- When there are no *empiric* rules (*shallow knowledge*) available to solve the problem, the system can fall back on *deep knowledge* and try to find a solution. Moreover, after a *deep* reasoning sequence a new rule, a *shallow knowledge*, may be extracted. This new rule may be used in cases where a similar situation occurs again (see [Van de Velde 86]).
- To provide good explanations.

4 Knowledge acquisition in Lézard

4.1 Basic principles

In the LÉZARD system we provide a way for acquiring *deep knowledge* and for extracting complete *shallow knowledge* before inferences. This knowledge acquisition process is performed in two steps:

Deep and shallow knowledge acquisition
The high level knowledge representation language allows one to describe *deep* models, and to represent *shallow knowledge*.

New shallow knowledge acquisition from deep knowledge
The system extracts production rules from *deep knowledge*. These rules are *shallow knowledge*. The extraction process is lead by *meta-knowledge* in interaction with the expert. The *meta-knowledge* is currently expressed in LISP-code. However, expressing the *meta-knowledge* in a more declarative manner seems to be feasible.

During this process knowledge is completed. For example, the expert can specify the reliability of the generated rules. Whatever the way this reliability is represented, it is necessary to model uncertain reasoning.

The extraction process may be broken down into three basic mechanisms : (i) formal derivation; (ii) semantic network interpretation; and, (iii) abduction. These three mechanisms are presented in the three following paragraphs.

4.2 Formal derivation

How can the knowledge C_1: "A AND B are-the S", which can be written (S = (OR A B)) be used?

The classic way to use this kind of knowledge is propositional calculus. A second, equivalent way is to transform C_1 into the 7 following production rules:

R_1: $((S \wedge (not\ A)) \Rightarrow B)$

R_2: $((S \wedge (not\ B)) \Rightarrow A)$

R_3: $(((not\ A) \wedge (not\ B)) \Rightarrow (not\ S))$

R_4: $A \Rightarrow S$

R_5: $B \Rightarrow S$

R_6: $((not\ S) \Rightarrow (not\ A))$

R_7: $((not\ S) \Rightarrow (not\ B))$

and then to use an inference engine. Our method is close to the latter way.

However, we are dealing with uncertainty. Propositions, for example: C_1 shown above, are believed to be, or are generally true, but are not certain. The strict formal derivation (mapping C_1 to (AND R_1 R_2 R_3 R_4 R_5 R_6 R_7)), as written above, is not possible. The same is true for the propositional calculus. In order to illustrate this purpose, let us give a significance to C_1, A, B, S and R_1:

S signifies: flip a coin three times consecutively.
A signifies: in the three flips, one will observe at least one head and one tail.
B signifies: one observes heads three times.

The knowledge, C_1, is "When flipping a coin three times consecutively, one observes at least one head." C_1 can be considered acceptable (it is true seven times for eight). However, the rule R_1: "If one flips a coin three times consecutively, and if one observes three identical results, then one observes three heads," is inacceptable (it is only true one time for two).

The solution we adopt is to have the expert intervene in the formal derivation so as to avoid possible errors and to specify the knowledge. For example : when C_1 is encountered, the rule R_1, among other rules, is generated, and the user is asked about its validity (or its reliability). When it is valid, or sufficiently reliable, it is conserved and put in the *shallow knowledge* base. In this case the user is also asked about the R_1's reliability level. In the opposite case, R_1 is discarded.

Among possible derivations, it is necessary to select the valid one. In this case we must ask the expert to validate a derivated knowledge. But the questions asked to the expert should be opportune: the foundations of the interpretation mechanisms which generate the rules must be elicited.

A certain number of recent works share a common preoccupation which can be described by the following sentence: "Q may not be believed, even though P is believed and (P $\Rightarrow$ Q) is believed". These formal systems are called "*logics of belief*". In the logic L of Levesque [Levesque 84, Charlet & Gascuel 86] a distinction is made between *implicit* and *explicit belief*. *Implicit belief* is possible belief. for example: Q in the example above. It obeys to all the classic rules of derivation. *Explicit belief* follows much more limited derivation rules. For example one can explicitly believe (OR A B) without explicitly believing A or explicitly believing B. However, if one explicitly believes (AND A B), then one explicitly believes A and explicitly believes B.

This distinction between beliefs lends itself perfectly to our system. When mapping from KB1 to KB2, some derivations are admissible with respect to explicit belief and, in this case, TRANSLATOR does not ask the user any questions about the mapping. Other derivations are only admissible with respect to implicit belief, and, in this situation, TRANSLATOR asks the user whether the derivations are valid (see the example given above). Our set of derivation rules for

explicit belief includes all the rules used by Levesque, plus a certain number of rules specific to the semantics of KB1.

In the LÉZARD system a similar reflexion to Levesque's is implemented. But instead of dealing with the logic L in the inference process, we propose to take it into account sooner in the model interpretation. Thus the system will not have to handle this task in the inference process.

4.3 The semantic network interpretation

Like several other diagnostic systems, and particularly medical diagnostic systems (see ANGINA [Long 82] or ABEL [Patil 82]), LÉZARD system is based on the hypothesis of "the unique fault process." According to this hypothesis one considers that multiple disorders (in medicine: multiple diagnoses) may occur simultaneously if and only if they are connected by causal relationships.

This hypothesis makes sense from a medical point of view. It is the first one made by a physician faced with a patient. Exceptions arise with patients whose general states are very deficient, e.g. a very old person, and who suffer from several independent disorders.

This hypothesis makes sense in other domains as well. At this level, the reasoning of a garage mechanic is most likely not very different from a physician's reasoning. This hypothesis may be seen as a *heuristic strategy* which is efficient in most practical cases. Remaining (complex) cases need more complex hypothesis. They will not be treated here.

According to this hypothesis, the system interprets the semantic network and proposes *mutual-exclusion* (versus *logical-compatibility*) between facts represented in the network. It therefor turns the *deep knowledge* into *shallow knowledge*.

Before explaining the principle of this interpretation, let us give some definitions:

- "X_1 :::> X_2" means that X_2 may be caused by X_1.
- A causal chain between the facts X_1 and X_2 is a chain of causal relationships between X_1 and X_2. For example: according to the semantic network given in the figure 2, there is a causal chain between APPENDICITIS and FUNCTIONAL-OBSTRUCTION:

  APPENDICITIS :::> PERITONITIS :::> FUNCTIONAL-OBSTRUCTION.

- A causal chain between the facts X_1 and X_2 becomes false when an intermediate fact in the chain becomes false, e.g. when PERITONITIS in the preceding example becomes false.
- Two facts X_1 and X_2 have a common cause Y when there is a causal chain between Y and X_1, and between Y and X_2. For example: according to the semantic network given in the figure 2, PERITONITIS is the common cause of ORGANIC-OBSTRUCTION and FUNCTIONAL-OBSTRUCTION:

  ORGANIC-OBSTRUCTION <::: PERITONITIS :::> FUNCTIONAL-OBSTRUCTION.

- A common cause between two facts, X_1 and X_2, becomes false when either this cause becomes false, or an intermediate fact in one of the two chains becomes false, e.g. when PERITONITIS becomes false in the preceding example.
- A fact X_2 is a descendant of a fact X_1 if X_1 is a father of X_2 in a taxonomy, e.g. APPENDICITIS is a descendant of DIGESTIVE-DISEASE.

- Two diagnoses are hierarchically linked if one is a descendant of the other, e.g. URINARY-DISEASE and RENAL-COLIC are hierarchically linked.
- Two diagnoses X_1 and X_2 have a common descendant if there is a third diagnosis Y which is a descendant of X_1 and X_2, e.g. DIGESTIVE-DISEASE and OBSTRUCTION have a common descendant which is ORGANIC-OBSTRUCTION.

Two diagnoses X and Y are *logically incompatible* if they satisfy the 4 following criteria:

- X and Y are not linked by a causal chain and they do not have any descendants linked by a causal chain (c1),
- X and Y do not have a common cause and they do not have any descendants having a common cause (c2),
- X and Y are not hierarchically linked (c3),
- X and Y do not have a common descendant (c4).

On the other hand two diagnoses X and Y are *non-trivially logically compatible* if they satisfy the following criteria:

- X and Y are not hierarchically linked (c3), (otherwise they would be *trivially logically compatible*),
- X and Y do not have a common descendant (c4) (otherwise they would be *trivially logically compatible*),
- X and Y or their descendants are linked by a causal chain, and this chain is not reduced to a single causal relationship, or X and Y or their descendants have a common cause (otherwise they would be *trivially logically compatible*) (c5).

Conditional logical incompatibility — or *conditional mutual exclusion* — is defined as follows:

- X and Y are not hierarchically linked (c3),
- X and Y do not a have common descendant (c4),
- X and Y are *conditionally logically incompatible* when the link between them (causal chain or common cause) becomes false (c6).

According to these principles, the system generates the following *mutual exclusion*:

DIGESTIVE-DISEASE and URINARY-DISEASE are mutually exclusive
VASCULAR-DISEASE and URINARY-DISEASE are mutually exclusive
OBSTRUCTION and URINARY-TRACT-INFECTION are mutually exclusive
RENAL-COLIC and URINARY-TRACT-INFECTION are mutually exclusive
ORGANIC-OBSTRUCTION and APPENDICITIS are mutually exclusive
PERITONITIS and URINARY-DISEASE are mutually exclusive
VASCULAR-DISEASE and APPENDICITIS are mutually exclusive
OBSTRUCTION and URINARY-DISEASE are mutually exclusive.

The generated *logical compatibility* are:

APPENDICITIS and OBSTRUCTION are compatible
VASCULAR-DISEASE and FUNCTIONAL-OBSTRUCTION are compatible
ORGANIC-OBSTRUCTION and FUNCTIONAL-OBSTRUCTION are compatible
DIGESTIVE-DISEASE and FUNCTIONAL-OBSTRUCTION are compatible.

The generated *conditional mutual exclusion* are:

APPENDICITIS and OBSTRUCTION are exclusive if PERITONITIS is false
VASCULAR-DISEASE and ORGANIC-OBSTRUCTION are exclusive if MESENTERIC-INFARCTUS is false
VASCULAR-DISEASE and PERITONITIS are exclusive if MESENTERIC-INFARCTUS is false
VASCULAR-DISEASE and FUNCTIONAL-OBSTRUCTION are exclusive if PERITONITIS or MESENTERIC-INFARCTUS is false.

The preceding knowledge is generated at the highest abstraction level. For example, the following:

DIGESTIVE-DISEASE and URINARY-DISEASE are exclusive

is preferred to the two pieces of knowledge:

DIGESTIVE-DISEASE and RENAL-COLIC are exclusive

DIGESTIVE-DISEASE and URINARY-TRACT-INFECTION are exclusive.

The expert is asked to confirm the generated knowledge. He thus chooses to accept it or not in the *shallow knowledge* base.

Mutual exclusions, i.e. A_i and B are mutually exclusive, are translated in production rules, i.e. "$A_i \rightarrow$ not B" and "B $\rightarrow$ not A_i".

Conditional mutual exclusions, i.e. A_i and B are mutually exclusive if C is false, are translated in production rules, i.e. "A_i and not C $\rightarrow$ not B" and "B and not C $\rightarrow$ not A_i".

Logical compatibilities are not translated into production rules. They are proposed to the expert in order to allow him to check the deep model validity.

The figure 3 gives an extract of a possible dialogue between LÉZARD and an expert. This dialogue concerns the interpretation of the network represented in figure 2.

4.4 Abduction

Abduction proposes a different logical interpretation of the causal relationships. It is acknowledged to be useful in medicine [Fieschi 81] and more generally in diagnostic tasks. The principle is as follows: "if I know that Q is true and Q is caused by P, then I can suppose that P is true". The abduction transforms "Q caused by P" into "Q $\Rightarrow$ P".

For example, if the system knows that PERITONITIS may be caused by APPENDICITIS, it proposes to the expert the following rule: "PERITONITIS $\Rightarrow$ APPENDICITIS".

The implicit purpose consists in asking the expert if PERITONITIS has always APPENDICITIS as a cause.

```
Do you want an exclusion relationship :
DIGESTIVE-DISEASE and URINARY-DISEASE are-exclusive (y/n) ? y
Do you want an exclusion relationship :
VASCULARY-DISEASE and URINARY-DISEASE are-exclusive (y/n) ? y
Do you want an exclusion relationship :
OBSTRUCTION and DISEASE URINAIRE-BASSE are-exclusive (y/n) ? y
Do you want an exclusion relationship :
RENAL-COLIC and URINARY-TRACT-INFECTION are-exclusive (y/n) ? y
Do you want an exclusion relationship :
VASCULARY-DISEASE and APPENDICITIS are-exclusive (y/n) ? n
Do you want an exclusion relationship :
OBSTRUCTION and URINARY-DISEASE are-exclusive (y/n) ? y

Are you aggre with the fact
APPENDICITIS and OBSTRUCTION are compatible (y/n) ? y
Are you aggre with the fact
VASCULARY-DISEASE and FUNCTIONAL-OBSTRUCTION are compatible (y/n) ? y

Are you aggre with the fact
ORGANIC-OBSTRUCTION and FUNCTIONAL-OBSTRUCTION are compatible (y/n) ? n
Attention. To refuse this compatibility is equivalent to refuse the model
or the interpretation of the causal relationship.
(1) You refuse the model, (2) you refuse the generated relationship (1/2) ? 1
Stop knowledge acquisition phase. You must modify the model
before revive the process.

≫
```

Figure 3: Dialogue between LÉZARD system and the expert about *mutual exclusion* and *logical compatibility* generated relationships.

This principle is also applied to implication relationships. For example, if the system knows that FEVER is a property of INFLAMMATION (or: INFLAMMATION $\Rightarrow$ FEVER), then it proposes to the expert the following rule: "FEVER $\Rightarrow$ INFLAMMATION".

The implicit purpose consists in asking the expert if FEVER is always a sign of INFLAMMATION.

5 Conclusion

The knowledge acquisition process of the LÉZARD system is interesting in several aspects:

- The details of knowledge interpretation must be fully specified when using an inference engine. Usually *deep knowledge* has a complex semantic which is not fully reflected by the used formalisms which have several possible interpretations themselves. In LÉZARD, the *deep knowledge* translation into *shallow knowledge* enforces to keep only those interpretations which are correct. This *shallow knowledge* is then treated without ambiguities by the

program while allowing a better control.

- The *deep knowledge* is completed in a precise manner by the shallow knowledge. The "interactive formal derivation" (see part 4.2) which is used for realizing this completion, gives a satisfactory solution to the problem of uncertainty.
- The *shallow knowledge* generation as a prior step to inferences allows the distinction between the processes. One can easily modify the deep knowledge representation and the translator functionalities without affecting the inference engine.
- This knowledge acquisition process is fast and efficient.
- The basic entity of shallow representation as production rules and the compilation of these rules make the engine run fast.

However, this knowledge acquisition process implies choices and constraints that must not be ignored:

- Although only the *shallow knowledge* is needed for making inferences, links between *deep knowledge* and *shallow knowledge* must be saved. Otherwise, the system's ability to explain will be very low. This is not currently implemented in LÉZARD. This constitutes an interesting direction for further research.
- There is no formal equivalence between the *deep knowledge* and its translation. However, in practice, this does not seem to be a great flaw. Knowledge is most often redundant. There usually exists several possible reasonings (several rules or combination of rules) for demonstrating the same conclusion.

6 Bibliography

Chandrasekaran B., Mittal S., 83. Deep versus compiled knowledge approaches to diagnostic problem solving. *Int. J. Man-Machine Studies*, 19:1983 (425–436).

Charlet J., Gascuel O., 86. Anatomie du système LÉZARD: le traducteur. In: *Proceedings of the 6th International Workshop on Expert Systems and their Applications, Avignon*, Gerfau, Paris, 1986;95–106.

Charlet J., Gascuel O., 87. Acquisition des connaissances dans le système LÉZARD. In: *Actes du colloque IA/Santé associé au SITEF, Toulouse*, Dubar, Toulouse, 1987;105–114.

Charlet J., 89. LÉZARD : acquisition des connaissances et gestion de l'incertitude dans un système expert de seconde génération. Thèse d'université, université Paris VI, 1989.

R. Cohen and E.A. Feigenbaum (eds), 82 Learning and Inductive Inference. In: R. Cohen and E.A. Feigenbaum (eds.), *The Handbook of Artificial Intelligence, vol 3*, W. Kaufmann, Inc., Los Altos, California, 1982;Chapter XIV.

Doyle J., 79. A Truth Maintenance System. *Artificial Intelligence* 12 (231–272) 1979.

Ferrand P., 84. SESAM: an Explanatory Medical Aid System. In: T. O'shea (ed.), *Proceedings of the 6th ECAI, Pisa*, North-Holland, Amsterdam, 1984;13–20.

Fieschi M., 84. *Intelligence artificielle en médecine. Des systèmes experts*, Masson, Paris, 1984.

Fink P.K., 87. Control and integration of diverse knowledge in a diagnostic expert system. In: A. Joshi (ed.), *Proceedings of the 9th IJCAI, Los Angeles*, M. Kaufmann, Inc., Los Altos, California, 1985;426–431.

Gascuel O., 85
Un système expert pour la réalisation de diagnostics.
Technique et Science Informatiques, 4(4):359–372 (1985).

Gascuel O., Charlet J., Genetet S., Mari B., 85
Le système LÉZARD un travail expérimental sur le raisonnement incertain.
In: *Proceedings of the 5th International Workshop on Expert Systems and their Applications, Avignon*, Gerfau, Paris, 1985;.

Klein, D., Finin, T., 87. What's in a Deep Model ? A Characterization of Knowledge Depth in Intelligence Safety Systems. In: J. McDermott (ed.), *Proceedings of the 10th IJCAI, Milan*, M. Kaufmann, Inc., Los Altos, California, 1987;559–562.

Konolige K., 79. An Inference Net Compiler for the PROSPECTOR rule-based Consultation System. In: *Proceedings of the 6th IJCAI, Tokyo*, M. Kaufmann, Inc., Los Altos, California, 1979;487–489.

Koton P.A., 85. Empirical and Model-Based Reasoning in Expert Systems. In: A. Joshi (ed.), *Proceedings of the 9th IJCAI, Los Angeles*, M. Kaufmann, Inc., Los Altos, California, 1985;297–299.

Kulikowsky C.A., Weiss S.M., 82 Representation of Expert Knowledge for Consultation: the CASNET and EXPERT Projects. In: P. Szolovits (ed.), *Artificial Intelligence in Medicine*, Westview Press, Inc., Boulder, Colorado, 1982;Chapter 2.

Kulikowski C., 87. Domain knowledge. In: S.C. Shapiro (ed.), *Encyclopedia of Artificial Intelligence, vol 1*, J. Wiley & Sons, Inc., New York, 1987;251–253.

Levesque H.J., 84. A Logic of Implicit and Explicit Belief. In: *Proceedings of the 4th AAAI, Austin, Texas*, M. Kaufmann, Inc., Los Altos, California, 1984;198–202.

Long W., Naimi S., Criscitiello M.G., Pauker S.G., Szolovits P., 84 An Aid to Physiological Reasoning in the Management of Cardiovascular Disease. In: *Proceedings of the Computers in Cardiology Conference*, 1984.

Mostow D.J., 83 Machine Transformation of Advice into a Search Procedure. In: R.S. Michalski, J.G. Carbonell and T.M. Mitchell (eds.), *Machine Learning, An Artificial Intelligence Approach*, Tioga Publishing Company, Palo Alto, California, 1983;367–403.

Patil R.S., Szolovits P., Schwartz W.B., 82. Modeling Knowledge of the Patient in Acid-Base and Electrolyte Disorders. In: P. Szolovits (ed.), *Artificial Intelligence in Medicine*, Westview Press, Inc., Boulder, Colorado, 1982;Chapter 6.

Politakis P., Weiss S.M., 84. Using Empirical Analysis to Refine Expert System Knowledge Bases. *Artificial Intelligence* **22** (1984) 23–48.

Shortliffe E.H., 76. *Computer-Based Medical Consultations: MYCIN*, Elsevier, New York, 1976.

Steels L., Van de Velde W., 85. Learning in Second Generation Expert System. In: J. S. Kowalik (ed.), *Knowledge Based Problem Solving*, Prentice-Hall, Inc., New Jersey, 1985.

Szolovits P., Patil R.S., Schwartz W.B., 88. Artificial Intelligence in Medical Diagnostic. *Annals of Internal Medicine* **108** (1988) 80–87.

Van de Velde W., 86
Learning Heuristics in Second Generation Expert Systems.
In: *Proceedings of the 6th International Workshop on Expert Systems and their Applications, Avignon*, Gerfau, Paris, 1986;883–899.

METAMETH

METHODOLOGY FOR KNOWLEDGE ENGINEERING

Inge Nordbø, Mette Vestli, Ingeborg Sølvberg
Knowledge Engineering Laboratory
ELAB-RUNIT
SINTEF
7034 Trondheim, Norway
E-mail: nordbo%vax.runit.unit.uninett@tor.nta.no

Abstract

METAMETH is a methodology for knowledge engineering developed within the METAKREK project conducted at SINTEF. The methodology emphasize the importance of knowledge modelling and propose modelling at different levels. The paper describes the modelling approach applied on an offshore application. Experiences from using the methodology in real world applications are described.

1 Introduction

Knowledge acquisition is the process of eliciting knowledge from different sources (usually a human expert) and modelling it in a computer in order to use the knowledge in problem solving. The METAKREK project [1] aims to improve the efficiency of the knowledge acquisition process by improving both methods and tools for development of knowledgebased systems ([SØLV88], [AAKV88]) .

Knowledge acquisition is regarded as one of the main problems requiring solution in order to achieve more competent knowledgebased systems. This is due to a number of problems (described in [WEL883]) which make development of knowledgebased systems more difficult than development of more traditional computer systems. Firstly, the models developed in software engineering are not directly applicable to the modelling of all possible kinds of knowledge. Secondly, expert tasks are usually too complicated to allow completion of a requirements specification in advance; this means that a more difficult iterative development and validation process is required. Thirdly, knowledge often changes very rapidly in an expert task, leading to more complicated maintenance problems.

According to [ANDE85], [ERIC85] and others, experts often retain parts of their knowledge as implicit "instinctive" knowledge through years of experience. In terms of knowledge elicitation this is a problem, because the expert is not able to verbalize the implicit knowledge, and the corresponding model of the domain in the knowledgebase will not be complete. In addition, there are problems with communication between expert and knowledge engineer, time consuming methods and constraints in implementation tools.

Much work is done on knowledge elicitation and how to get the knowledge from the expert. The results of these techniques are often written transcript of

---

[1] METAKREK - Methods for Acquisition and Representation of Knowledge is funded by NTNF - IT.0225.16935. The project ran from 1986 to 1988 with a total effort of 5.5 man-years. In addition a number of students have participated in the research by doing project work and master theses. A Ph.D. in machine learning is conducted within the project

the knowledge. An important problem is however how to use this knowledge to implement an expert system, there is a gap between the elicited knowledge and the implemented system. In order to fill this gap we need methods for intermediate representations - modelling.

Contemporary Knowledge Acquisition Research

Previous and ongoing research attack the knowledge acquisition problem from various angles. Development projects often concentrate on only parts of the problem.

Gammack and Young [GAMM84] argue that the shortcommings of current techniques are due primarily to two factors. Firstly, expertise in a technical domain comprises knowledge of more than one kind, not all of which can reasonably be represented in the form of empirical rules. Secondly, not all of these knowledge types are suitable for elicitation by (more or less) informal interview. [GAMM84] argue that an appropriate way to elicit knowledge in even a single domain is to use a variety of techniques, matched to the different kinds of knowledge in the domain. They discuss interview, protocol analysis, multidimensional scaling and concept sorting.

Breuker [BREU87] focuses on modelling of problem solving in KADS. A central idea in KADS is the use of on-line libraries of generic models of tasks and domains at the analysis stage. Others have researched into the modelling of problem solving strategies for specific tasks. Several research projects within modelling of diagnosis are described in [MILN87]; a particular strategy for diagnosis is proposed in [KERA86] and work on classification problems is described in [CLAN85]. [JOHN87] reports on an interesting approach to intermediate knowledge representation, the SGN-formalism (Systemic Grammar Network).

Work towards an overall methodology for developing knowledgebased systems has been done in some projects in recent years. KADS ([BREU87], [GREE85], [BREU85]) is a comprehensive methodology for knowledge acquisition. Within this project, a life cycle model for structured analysis of expertise was developed ([HAYW87]) The central part of the methodology is a detailed description of the architecture of a prospective expert system. The methodology is supported by tools for editing, storing and retrieving documents that are defined in KADS. Life cycle models for knowledgebased systems are also described in [ESA86] and [GROV83].

At different stages of the knowledge acquisition process, the use of machine learning is likely to improve efficiency. Most of the research in this area has been done within inductive learning ([QUIN83], [MICH83], [LEBO87], [STEP86]) and explanation-based learning ([MITC86]). In recent years, interesting work has been done within case-based learning / case-based reasoning [BARE87].

METAKREK

The METAKREK project aims to make the knowledge acquisition process more efficient by combining previous knowledge acquisition research, psychological methods and computer support. To achieve this, a methodology for knowledge acquisition (METAMETH) has been developed. METAMETH supports

- Methods and techniques for elicitation of different kinds of knowledge.

- Methods for knowledge modelling on different knowledge levels and a methodology for knowledge engineering that integrates methods and computer tools into a framework for knowledge acquisition.

A tool-kit, METATOOL, is developed to aid the methodology. It consists of different special purpose editors for knowledge editing (METABUILD) and a reasoning/explanation tool (METASHOW). Both editors and reasoning tools are supported with extensive grahics in order to enhance the user friendliness of the tool-kit and to support the creative knowledge elicitation process with the expert.

The paper gives a brief overview of the methodology with focus on modelling. Section 2 gives a short introduction to the methodology. The next section describes the knowledge modelling stage in more detail followed by an example. Section 4 gives a list of experiences drawn from the project and the last section identifies some further research lines.

2 Knowledge Engineering

In order to develop a knowledgebased system in a real world application, it is necessary to have a knowledge engineering methodology. This methodology should make the knowledge acquisition process more efficient, put structure upon the process and provide a set of optional methods and techniques (with guidelines on how to choose between them).

To achieve these goals, we have developed a knowledge engineering methodology which has been tested on real world applications and gives support in the knowledge acquisition process.

One of the basic design principles behind the methodology is that we stress the importance of going through the knowledge acquisition process independently of implementation decisions, in order to avoid the 'knowledge mismatch' problem. Our solution to this is to make models instead of making direct implementations of the knowledge.

2.1 The METAMETH Knowledge Engineering Methodology

In figure 2-1 the different stages in the knowledge engineering methodology are shown. The stages are to some extent the same as in a traditional 'waterfall' software engineering methodology, but the figure illustrates the point that developing knowledgebased systems requires a more iterative style than traditional software engineering. In all stages except for Problem Definition, machine learning may be useful, but is not covered by this paper (see [BLIX88],[SØLV88]).

Stage 1: PROBLEM DEFINITION

The goal of this stage is to

- define the problem
- conduct a feasibility study
- give the project group an overview of the application domain and the domain of knowledge engineering.

The output from this stage is a problem specification and project plan.

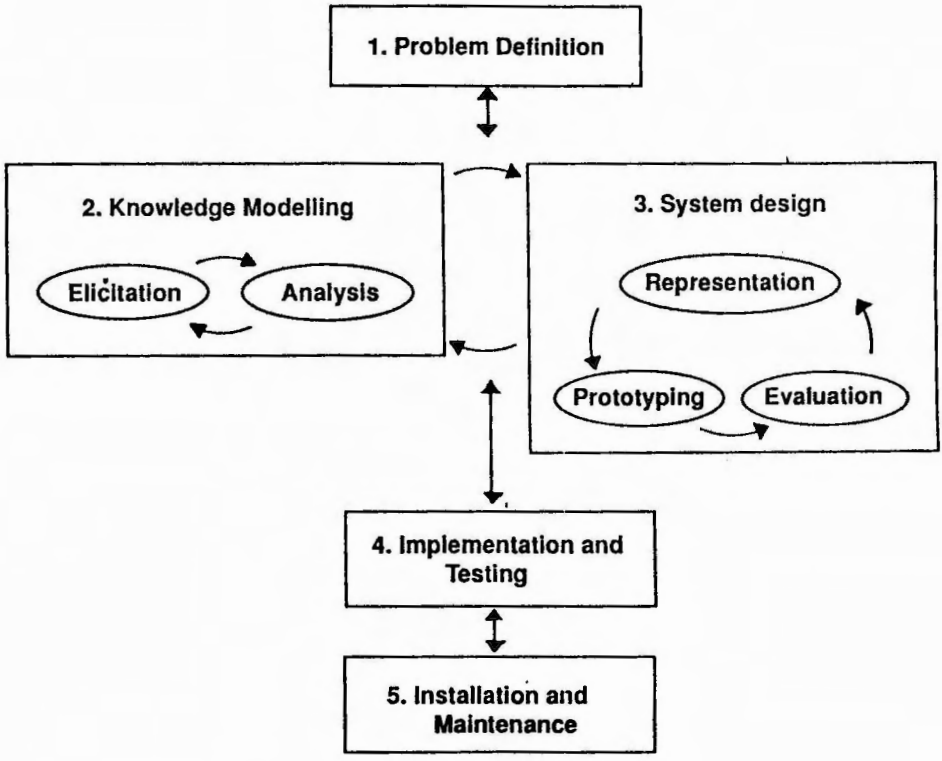


Fig. 2-1 Stages in the Knowledge Engineering Methodology

Stage 2: KNOWLEDGE MODELLING

The goal of this stage is to elicit knowledge and develop models of the domain independently of implementation decisions. The output of the stage is a set of models on which the system design is based.

The work in this stage is divided into two different activities with close interaction: elicitation and analysis.

Elicitation covers all the interaction with the expert in order to elicit the knowledge of the domain and produce a textual description of it. This knowledge is analysed in the analysis activity, resulting in an object model and a problem solving model (further described in section 3).

Stage 3: SYSTEM DESIGN

The activities in this stage should support the difficult problem of transforming models into running systems. The goal at this stage is to work out design principles for a successful implementation. This results in a number of decisions being taken regarding representation, reasoning, uncertainty, man machine interface etc. and in small prototype implementations of sub-models being produced for testing.

In the METAKREK project the METATOOL prototyping tool (described in [AAKV88]) has been developed which gives support in this stage. The tool combines frames and rules and it is supported with extensive graphics for visualizing both the knowledge bases and the explanation graphs of the reasoning. This tool also supports knowledge acquisition by interacting graphically with the experts.

Stage 4: IMPLEMENTATION AND TESTING

From the design principles and the problem specification an implementation tool is chosen. The system is implemented and tested as in traditional software engineering. During implementation and testing revisions of the models are often needed as the work progresses.

Stage 5: INSTALLATION AND MAINTENANCE

After the system has been installed at the user's site the maintenance process begins and continues throughout the life-time of the system.

3 Knowledge modelling

Knowledge modelling is essential for success in the development of expert systems. In our work we focus on modelling as an intermediate representation between the knowledge elicited from the experts and the knowledge implemented as a system. The models are used as a communication medium between experts and knowledge engineers, as an aid in structuring and describing the domain independently of the implementation and as an aid in making decisions about representation, reasoning strategies etc.

In our approach the modelling stage consists of elicitation and analysis. In this paper we concentrate on analysis. Knowledge analysis is the activity which analyzes the knowledge from the knowledge elicitation activity in order to structure it and develop models of the domain. There is a gap between the knowledge expressed by the expert and the executable code. In order to bridge the gap, intermediate representations are used. We propose a two step approach from the transcript to the conceptual model as illustrated in figure 3-2.

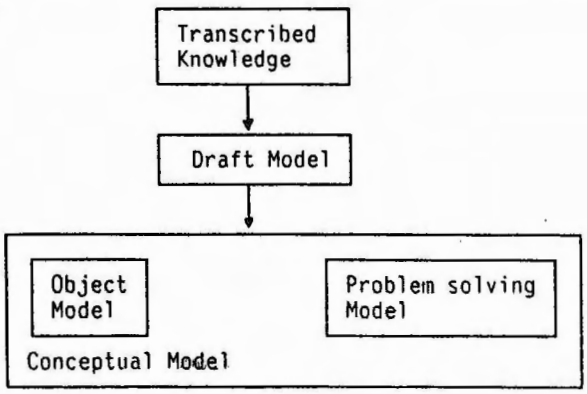


Fig.3-2 Two-step Knowledge Modelling

The conceptual model has two parts, an object model with the descriptions of the domain in terms of an object oriented formalism and a problem solving model focusing on the process of solving problems in the domain. These models are a more or less complete description serving as basis for further design. As an intermediate step we propose to use a draft model as a first structuring of the domain.

The rest of this section is used to demonstrate the modelling approach in METAMETH. The example gives overview of the formalisms we propose and the modelling steps from a transcript towards a prototype representation of the knowledge.

3.1 Knowledge modelling in mud diagnosis

In this chapter we use an example from a project developing a prototype expert system for Drilling Fluid Diagnosis (the application domain is similar to that of MUD [KAHN84]). The mud engineer has a set of mud parameters (viscosity, alkalinity, solid content etc.) from which he reads the status of the mud system. Problems could be contaminants from the formation or incorrect additives in the mud system.

In this example, we illustrate the modelling activity with a case where the problem is caused by a rocksalt contamination. Rocksalt changes both viscosity and alcalinity, which means that the properties of the mud are changed.

The starting point is part of a transcript from the elicitation activity where the elicitation method was a case review with the mud-engineer:

> Mud-engineer:
> The first thing to notice is that Funnel viscosity has increased. We check Yield point (YP) and find that it has increased, as has plastic viscosity (PV). Since YP has increased more than PV, I think something chemical has occured. To test this further, we check and find that Ph has increased, as has alkalinity, and that chlorides (Cl) have increased considerably.
>
> With such a strong increase in Cl, it could be rocksalt contamination or saltwater influx. To find out which of these is the problem, we check whether the volume of the mud system has increased, and whether the formation is known. In this case we know that the formation contains rocksalt, and rocksalt is the solution to this problem.
>
> With chlorides we must be aware of the danger of flocculation which 'opens' the filter-cake and fluid-loss could occur.
> ..

Fig.3-3 Example of Mud Transcript

DRAFT MODEL

The draft model is used as an aid for communication between experts and knowledge engineers, to ensure that the knowledge engineers have the necessary understanding of the domain. A description of the knowledge necessary to understand problem solving is also incorporated in the draft model from the

protocol transcripts. The draft model usually consists of a number of different 'submodels'.

In the mud-example, we developed the draft model in a formalism called SGN (Systemic Grammar Network) [JOHN87] with good results. The network is a graph composed of nodes joined with links. The network has basically one link, an and/or link denoted by curly/straight brackets. There are also other notions, i.e. a recursion symbol allowing optional choices. The SGN formalism provides a freedom of expression which allows the knowledge to be represented in a very natural way, rather than having to contort it to fit the representation technique.

The transcript was analysed by marking important concepts and relations and trying - in cooperation with the expert - to develop some kind of structured description. In this case we developed different sets of networks to view the problems from different angles. Shown in figure 3-4 are two diagrams, one shows which parameters will change if the contamination is rocksalt, the other diagram shows which contaminants will cause a change in Funnel Viscosity. As we developed diagrams focussing on different views of the domain, the expert could concentrate on one feature at a time and the different diagrams was used together to make a more complete description and to check the correctness of the information in the each diagram.

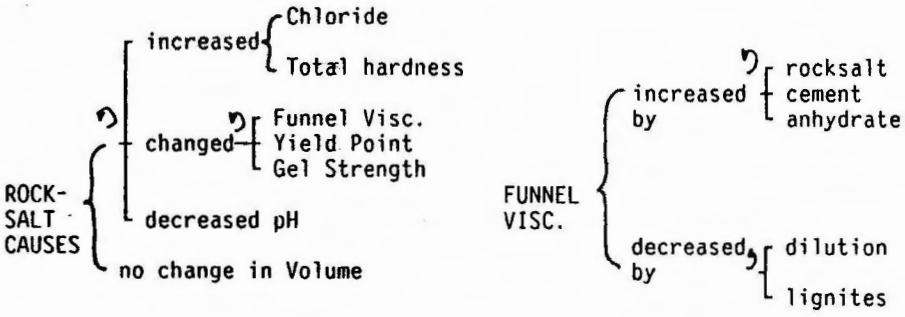


Fig.3-4 Draft model in SGN formalism

OBJECT MODEL

The object model is a comprehensive description of the domain which describes everything as an object, and provides a description of the domain which is as complete as possible. The proposed formalism for the object model is a combination of semantic networks and frames.

The object model is usually based on the draft model and more interviews with the experts. By using structured interviews focusing on parts of the model, extensions to the draft model (object descriptions, various relations, constraints, causal relationships etc.) are incorporated into the object model.

In the mud example we used the information in the draft model and merged the sub-models to a larger, more complete model expressing the objects, objects description and the relations. In figure 3-5 we show objects in the domain with structural and causal relations. An example of an object description is shown for Funnel Viscosity with two slots, value and value-change. The model

shows also that value-change can take one of three values: increased, decreased or normal.

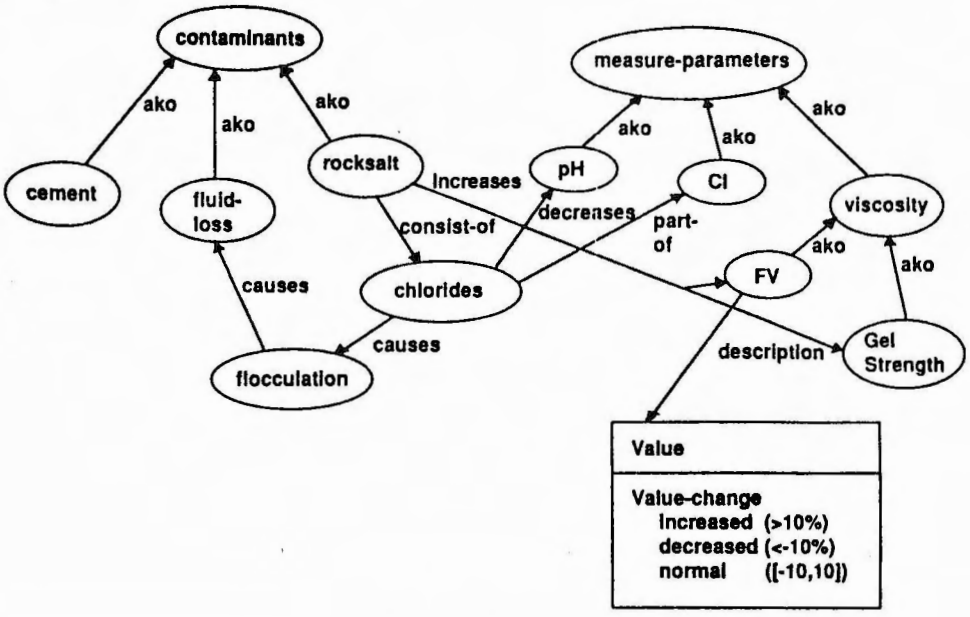


Fig. 3-5 Object model

PROBLEM SOLVING MODEL

Development of the problem solving model is based mainly on the understanding of the problem area gained from working out the draft model. The model is a structured description of a strategy for how to reach the right solution from an input description. The 'right' problem solving model can be used in development of the object model where the different processes and states are used to focus on different parts of the domain. Implementing the model in the final system will also support a more natural dialogue with the end-user.

An example of a problem solving model developed during analysis of protocols and discussions in the Mud application is shown in figure 3-6. The resulting model was adapted from a discussion of different ways of expressing problem solving strategy for diagnosis ([DURA88]). The model will be used as the overall inference strategy for using the knowledge in the object model to solve problems.

The figure expresses the following:
Mud parameters are gathered during the whole process. Observed and inferred values (e.g. a Funnel Viscosity increase of 10 % is a significant increase) will 'remind' the system of contaminants and additives that may be the cause of the problem. From this process the system makes a list of possible hypotheses (contaminants or additives) that is used to predict which mud values are expected to change. This list of hypothesis is also used to find which of the observed or inferred values are relevant to the different hypotheses. Based on a comparison of relevant observed and expected values, a conclusion is drawn.

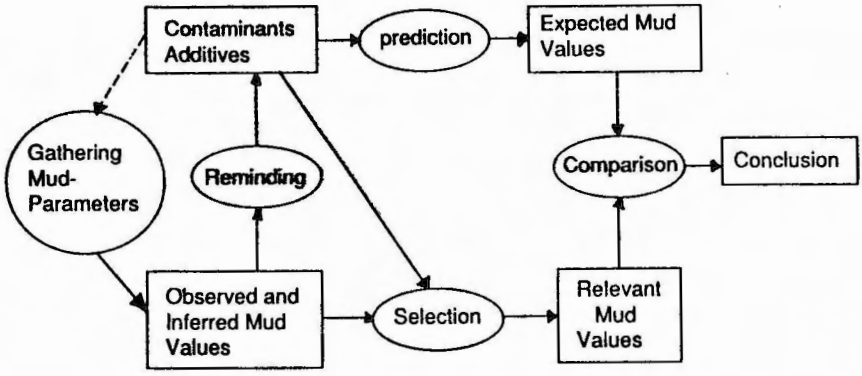


Fig. 3-6 Example of problem solving model

KNOWLEDGE REPRESENTATION WITH METATOOL

When the models are developed, the knowledge needs to be tested in order to correct errors and make extentions to the models. METATOOL is used in this application to test parts of the object model. The object description and the structural relations are expressed with METATOOL Structure Editor. A small part of the model is showed in figure 3-7 for illustration purposes.

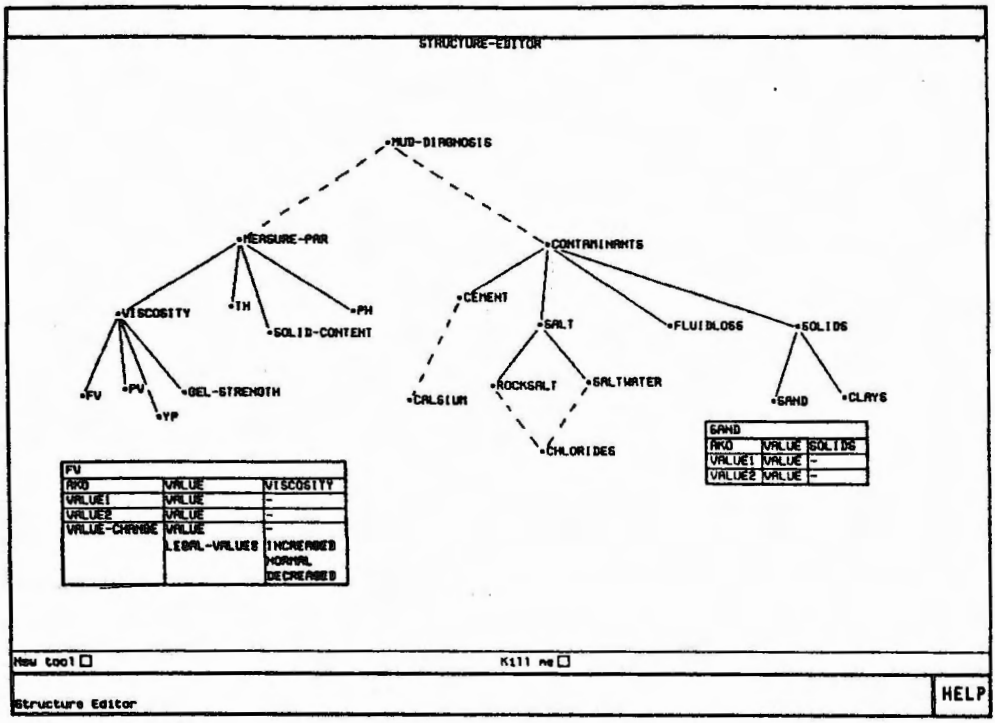


Fig. 3-7 Example of object representation in METATOOL

The other relations in the object model may be expressed in a rule language with METATOOL Rule Editor. Figure 3-8 shows some of the rulenames in the knowledge base. In order to make most of the knowledge explicit the conversion from numeric to symbolic values is also expressed as rules. (See FV-INCREASE. This rule says that a change in value of FV is termed "increased" if the value has increased by more than 10 percent from the last measurement.) The figure also shows an example of a rule for concluding that rocksalt causes the problem.

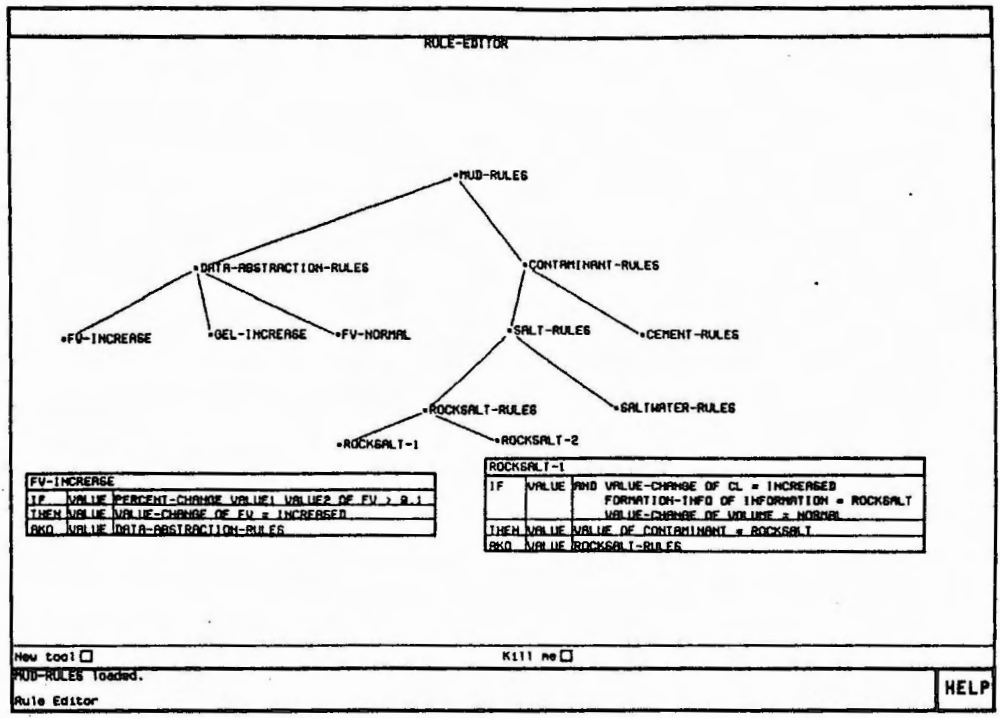


Fig. 3-8 Example of rule representation in METATOOL

4 Experiences

The methodology described in this paper was tested on two real world applications (Offshore: Drilling Fluid Diagnoses and Medicine: Interpretation of Serum protein electro phoreses [SØLV88]). The applications gave som interesting experiences as summarized below.

The methodology
The described methodology seems to make it possible to make a rather good model of an application domain before selecting a representation formalism and implementation tool. Experience from the application also supports the use of prototyping on a high level tool (f.ex. METATOOL) in order to refine the model and choose final representation techniques and implementation tools.

Our experience is that the methodology structures the work and makes the acquisition process more efficient. Since the methodology gives an outline of

the work and places different activities in time, the cooperation with experts are easier. In our case the experts agreed on postponing prototyping because prototyping was an activity in a later stage.

Modelling
To structure the domain in models instead of expressing knowledge in some kind of representation formalism has proved to be an efficient way of helping the expert express the knowledge unconstrained. Experiences from earlier projects where tool has been selected early, shows that experts are clever at expressing the knowledge in the particular formalism (i.e. rules) and avoiding knowledge that does not fit in. Experience from using models indicates that this problem is reduced.

SGN-formalism
Another experience from the work with the applications is that the SGN-modelling formalism is very helpful in the first part of the modelling stage. It is rather easy to draw a SGN-model from a written transcript, the model serve as a good basis for communication between experts and knowledge engineers and is suitable for expressing different views of the same knowledge. The experts also found it easy to draw diagrams on their own.

5 Further research

The research in the METAKREK project shows that the methodology gives valuable support with respect to modelling at different stages, but it can be improved by work on transforming between models of knowledge. More efficient tools to support the modelling part of the acquisition process are also necessary.

The methodology could also be improved in terms of modelling of problem solving strategy. The work done in KADS [BREU87] is an important contribution to this and should be developed further.

Another improvement would be to develop a more powerful modelling formalism for expressing objects, relations, constraints, heuristics etc. on a higher level of abstraction than current techniques for knowledge representation.

The field of knowledge engineering still lacks a comprehensive guide to support the whole development process of knowledgebased systems. A guide should describe methods, how to use them, when to use them, how to analyse the result, how to combine them, which formalisms to use, how to get from one stage to another etc.

Methods for automatic updating of the knowledge base during problem solving have now raised to a level of practical interest. Recently there has been a shift (or rather a split?) in machine learning research towards more knowledge intensive learning methods. Both explanation based and case based methods have so far been 'pure' systems, that is limited to a single approach, more or less. A very interesting direction to pursue would be to integrate case based methods with learning and reasoning techniques based on a strong explanation mechanism. This would require a tight coupling between the knowledge model of a domain (concepts, relations, constraints, etc.) and the features, hypotheses, expectations and solutions associated with past cases. The knowledge model could be used to constrain and direct the selection of relevant features, the case matching process, the evaluation of suggested solutions, etc.

The topics described above will, to a large extent, be topics for further research within SINTEF. The largest part of these activities will take place within the ACKnowledge project. ACKnowledge (ACquisition of Knowledge, Esprit-

project 2576) is aiming at developing KEW - Knowledge Engineering Workbench - and to develop guidelines of how to use and combine knowledge acquisition techniques. (Prime contractor is Cap Sesa Innovation.)

References

[ANDE85]: Anderson, J. R.: Cognitive Psychology and its Implications. (2. ed.) Freeman & Co, 1985.

[BARE87]: Bareiss, R., Porter, B., Wier, C.: PROTOS - an Examplar Based Learning Apprentice. In Proceedings of 2nd Knowledge Acquisition for Knowledge Based Systems Workshop Banff, Canada. 1987.

[BLIX88]: Blix, G.: Integration of Learning in Knowledge Acquisition. Trondheim 1987, 133 p. STF14 A88018, RIK-88-7, ISBN 82-595-5167-5.

[BREU85]: Breuker, J., Wielinga, B.: KADS: Structured Knowledge Acquisition for Expert Systems. Proceedings of the Fifth International Workshop on Expert Systems and their Applications, Avignon, 1985.

[BREU87]: Breuker, J. et al : Model driven knowledge acquisition: interpretation models. Deliverable task A1, Esprit project 1098, University of Amsterdam, 1987.

[CLAN85]: Clancey, W. J.: Heuristic Classification. Artificial Intelligence 27, pp. 289-350. 1985.

[DURA88]: Duran, R., Bareiss, R.: Personal discussions at the University of Texas at Austin. 1988

[ERIC85]: Ericsson, K.A., Simon, H.A.: Protocol analysis: Verbal Reports as Data. MIT-press, 1985.

[ESA86]: Life-Cycle for an Expert System ESA TMS/86-091/MG/md

[GAMM84]: Gammack, J. G., Young, R. M.: Psychological Techniques for Eliciting Expert Knowledge. Proceedings of 4. Technical Conference of the British Computer Society Specialists Group on Expert Systems, pp. 105-112. 1984.

[GREE85]: de Greef, P., Breuker, J.: A Case Study in Structured Knowledge Acquisition. Proc. of 9. IJCAI-85, pp. 390-392. 1985.

[GROV83]: Grover, Mark D.: A Pragmatic Knowledge Acquisition Methodology. Proc. of 8. IJCAI-83, pp. 436-438. 1983.

[HAYW87]: How to build knowledge-based systems:techniques, tools and case studies. In Commision of the European Communities, Directorate General Telecommunications, ESPRIT'87 Achievements and Impact, Elsevier Science Publishers B. V. 1987.

[JOHN87]: Johnson, L., Johnson, N.E.: Knowledge Elicitation Involving Teachback Interviewing. In Kidd: Knowledge Acquisition for Expert systems. A pratical handbook. Plenum. 1987.

[KAHN84]: Kahn, G., McDermott, J.: The MUD system. Proceedings of the First conference on AI applications. Denver, IEEE/AAAI. 1986.

[KERA86]: Keravnou, E.T, Johnson, L.: Competent Expert systems. Kogan page Ltd. 1986.

[LEBO87]: Lebowitz, M.: Experiments with Incremental Concept Formation: UNIMEM. Machine Learning 2, p 103-138. 1987.

[MICH83]: Michalski, R., Carbonell, J., Mitchell, T.: Machine Learning: An AI approach. Tioga Press. 1983.

[MILN87]: Milne, R.: Strategies for Diagnosis. IEEE Transactions on system man and cybernetics, Vol SMC-17, No 3. 1987.

[MITC86]: Mitchell, T., Keller, R., Kedar-Cabelli, S.: Explanation Based Generalization. Machine Learning 1, 1986.

[QUIN83]: Quinland J.: Learning Efficient Classification Procedures and their Application to Chess and Games. In (MICH83), 1983.

[STEP86]: Stepp, R., Michalski, R.: Conceptual Clustering: Inventing Goal-oriented Classification of Structured Objects. In Michalski et al: Machine Learning: an AI approach 2. 1986.

[SØLV88]: Sølvberg,I., Nordbø,I., Vestli,M., Aakvik,G., Amble,T., Eggen,J., Aamodt,A.: METAKREK-Methodology and toolkit for Knowledge Acquisition STF14 A88046, ISBN 82-595-5310-4, RIK88-9.

[VEST89]: Vestli,M., Sølvberg,I., Johnsen.H., Kjøbli,E., Eggen,J., Prytz,M., Romslo,I., Nordbø,T., Gleditsch, J.: METAKREK - A Knowledge Acquisition Case Study, Proceedings of 9. Int. workshop on Expert systems and their Applications, Avignon. 1989.

[WELB83]: Welbank, M.: A Review of Knowledge Elicitation Techniques for Expert systems. British Telecom Research Lab. 1983.

[AAKV88]: Aakvik, G., Vestli, M., Nordbø, I., Sølvberg, I., Aamodt,A.: METAKREK : METATOOL Documentation. STF14 A88055, ISBN 82-595-5315-5, RIK-88-12. Trondheim 1988.

Towards Knowledge Acquisition in Natural Language Dialogue

Gabriele Schmidt and Thomas Wetter

IBM Germany Scientific Center
Institute for Knowledge Based Systems
Knowledge and Language Processing 1

Abstract

When constructing and maintaining a knowledge base, the domain expert has to concentrate on the contents of the communicated knowledge items. The quality of his work may be affected, if he also has to pay very much attention to writing in a formal language and to checking a number of integrity conditions. We present the system KALEX, which allows the user to write knowledge items in natural language sentences (German). The system attempts to translate the sentences into its internal respresentation, performs some checks on the formal result of the translation, and allows immediate test of any transient state of the knowledge base with respect to already known or newly added cases to be solved.

The system has been successfully tested in a case study to extend a legal knowledge base to new variants of the concept of "damage".

Introduction

The acceptance and the confidence that is placed in an expert system by an expert, who is not at the same time a computer specialist, will grow by giving the expert the possibility to construct, test and maintain the system by himself/herself. Providing this possibility should not mean that the expert has to become a computer specialist [Tuhri88] but in most cases it still means that the domain expert uses (and hence has to learn) a high level **formal** language.

The expert, however, in many domains explains, argues, recalls, etc. in complete sentences or sentence fragments of his/her **native** language plus specific terms of his/her domain (cf. [Silve87; Lefko87; Diede87]). It may be asked, why there are only few attempts to communicate expert knowledge in natural language to the computer. The answer, so far, has been that computational linguistics has not yet reached a state, which allows the user a sufficient expressiveness to formulate what he/she knows or considers. This implies that the gap between the user's expectation about the system capacity and the real capacity is too large. Hence, the user might be more cognitively distracted and motivationally frustated (by having to seek for an expression that the "stupid system" understands) than supported by the facility to use his/her native instead of some formal language.

This is probably still the case now, but computational linguistics is approaching a level, where it might be useful for knowledge acquisition. This is exemplified by recent achievements reported in [Grosz87; Lehma85; Blase88], to give just a few general ones, and specific achievements for knowledge acquisition such as [Szpak88].

The aims of this paper are:

- to analyse the problems arising differently with natural language knowledge acquisition interfaces than with formal ones,
- to characterize domains, where knowledge is available in natural language,
- to describe the present state of an implementation of a natural language knowledge acquisition environment,
- to analyse to what extent the above problems are solved in the implementation and what else has to be done.

Problems in Natural Language Knowledge Acquisition

When using a natural language analyser for knowledge acquisition, one encounters certain natural language related problems, which become clearly evident when comparing a formal language interface with a natural language interface.

Syntax A formal language has some precisely defined syntax, which is implemented in the system and has to be newly learned by the user.

When using his/her native language, the user has a sound feeling of the set of correct sentences, which may, however, deviate from the set of sentences accepted by any existing grammar. I.e. the user will use sentences which cannot be analysed by the system.

Hence, measures have to be taken to close the gap between patterns spontaneously produced by the user and those accepted by the system.

Semantics A formal language also has (or should have) a semantics, which is well understood by the system designer but has to be learned and controlled by the user.

When using his/her native language, the user intends some meaning for any word, sentence etc. he/she uses. But as long as machine inference can only be conducted on (what so ever) formal expressions, any knowledge item has in the end to be represented in a formal language. And as long as there is no verified one-to-one correspondence between natural language expression and their formal images which reflect the intended meanings of all accepted sentences, natural language only serves as an interface. The interface may derive meanings from sentences which deviate from the intended meanings of the user. This includes the problem of technical terms with special meanings. So we need both, means to reduce the number of misunderstood sentences and means to repair concrete instances of misinterpretations.

Pragmatics In formal languages a distinction between object level knowledge and meta-level descriptive or control knowledge can only be expressed by the user, if the language offers specialized syntactic constructs for this purpose. Hence the user has no chance to communicate any type of meta-information that has not been anticipated by the system designer.

In natural language, on the other hand, it has been observed [Silve87; Wette88] that rules, results of applications of rules, justifications for rules, and other unrelated material occur unsystematically in expert utterances. To distinguish automatically between these pragmatic categories certainly goes beyond existing natural language analysers. Hence it has to be done manually on the basis of some domain model [Breuk87; Silve87] or non-rule sentences have to be avoided in advance by adequate design of the knowledge acquisition environment. This design will also have to take into account that it has been doubted that the expert has available the rules he/she applies.

Availability of Knowledge in Natural Language

There are some encouraging observations about general availability of object level rules in the expert and about measures to increase the amount of usable statements.

In domains where a major part of the knowledge is terminological or where the cases to be solved are naturally described in natural language we conducted experiments which revealed that - in contrast to many observations and conjectures indicating the implicitness of essential parts of the knowledge (cf. [Hayes85; Nisbe77]) - the domain expert has a relatively clear account of the terminological or ontological structure of his/her domain. Appendix 1 gives a typical example of a session, where a legal scientist spontaneously provided a top down elaboration on "theft". In the legal domain there is the additional advantage that the terminology used in such elaborations is highly normalized - no lawyer would use the common sense synonym "stealing"-. The top down manner of elaborating is closely related to the legal method of subsumption, i.e. proving norm conditions from case facts, which is mandatory in continental civil law systems.

One can conclude that for considerable (core) parts of domains such as law, natural language formulations are readily available for the expert, and hence, an operational natural language interface as the presented one considerably reduces the knowledge acquisition problem.

For domains where the knowledge is either less available or less coded in natural language (which peculiarly applies to common sense knowledge) other characteristics of the environment have also to be adjusted to the needs.

Concerning the distinction between (useful) object level information and other (useless) utterances, [Wette88] showed that by adjustment of some details in the setting of presenting cases and recording related utterances the amount of useful information could be highly increased.

The LEX1 Expert System Shell Environment for the Knowledge Acquisition Component

Knowledge Representation and Inferencing The target expert system shell for the natural language knowledge acquisition component has 1st order predicate logic as representation language. Inferencing is performed by a tableau calculus theorem prover (cf.[Schoe85]).

General Purpose Natural Language Analyser Complementary to this accomplishment in mathematically founded inferencing, several components for automated analysis of natural language have been implemented. We are using a dictionary of the 2000 most common German words, which is a subset of a sizable dictionary (ca. 20000 main category entries in an SQL database, cf. [Barne86]). There is a related rich grammar (extended from [Zoepp84]), which interfaces with semantic processors that use selection restrictions and meaning rules for resolving ambiguity and contextual reference. Texts are finally mapped on discourse representation structures (DRS's) i.e. essentially 1st order predicate logic formulae [Guent86] by construction rules that are based on Kamp's [Kamp81] Discourse Representation Theory. These components have been successfully used for natural language queries to a relational database [Lehma85], for analysis of narrative texts (traffic accidents), and for natural language dialogue about the (legal) contents of a text [Haft87].

Components of the Knowledge Acquisition System

We now describe a natural language knowledge acquisition system KALEX (Knowledge Acquisition for LEX), whose properties are discussed in the following paragraph as partial solutions to the problems arising with the use of natural language[1] .

Natural Language User Interface The user enters a knowledge acquisition session by writing a rule sentence and requests the system to analyse it and translate it into a DRS.

In the case of unrecognized syntax he/she receives a message and can derive from a list of related formulations, how he/she might modify his/her sentence in order to make it analysable for KALEX.

In those cases, where unknown words in certain places in a sentence are the only formal deficiency of a sentence, and in the favorable cases of syntactically correct (in the sense of the implemented grammar) sentences, KALEX presents both to the user, the input sentence and the automatically produced DRS in two windows of a screen (for details of DRS syntax and semantics cf. [Guent86]).

In the first example, a complete parse is achieved.

---

[1] A detailed description of the implementation of KALEX can be found in [Schmi89].

1 ***Every human, who walks, is a pedestrian.***

```
1  for all e1,u1,t1
2     human(u1) &
3     walk(u1,e1) &
4     time_of(e1,t1)      ->     pedestrian(u1).
```

In this case the DRS supplied by KALEX is completely correct, i.e. the user has no need to change. In some sense this also refers to the following example[2]:

2a ***Jede Landstraße oder Bundesstraße ist eine Straße.***

```
1   (for all u1
2      (for all u2,u3
3         'Landstraße'(u1,u2,u3))  !
4      (for all u4,u5,
5         'Bundesstraße'(u1,u4,u5)))
6    -> (exists u6,u7,
7         'Straße'(u1,u6,u7)).
```

Using this DRS enables certain proofs, where *Straße* is part of the proof goal and e.g. ***Bundesstraße*** is known. Of course this could be extended to include simple terms such as ***Autobahn*** or complex descriptions of other streets. A closer look at the meaning of the variables ***u1 through u7*** reveals that a more specific version, i.e. one which further restricts unification, is more adequate and still enables the desired proofs. The three variables, say ***u1, u2, u3*** of the predicate *Straße* denote ***Straße u1 von u2 nach u3*** (***street u1 from u2 to u3***). Hence, the automatically generated DRS allows the premises to have identical *or* different from/to substitutions, while the user's intended meaning here is to have identical substitutions[3].

2b

```
1   (for all u1,u2,u3
2      'Landstraße'(u1,u2,u3)  !
3      'Bundesstraße'(u1,u2,u3))
4    ->
5      'Straße'(u1,u2,u3).
```

To arrive at this intended representation, the user can make use of the knowledge editing facility described below.

Editing Knowledge Elements For working over a system suggestion, the user selects the editing environment, which widely resembles the former environment. This environment has been designed as to permanently remind the user of the aim he/she pursues by presenting him/her his/her original natural language sentence(s) (in one window) and to enable editing on DRS patterns, which should come close to his/her intended formal representation (in a second window).

---

[2] The whole system is operational in German. Only the grammar and dictionary have also been written in English, French, Italian, Spanish and Danish. Since much of the syntax and semantics is different in other languages than German, we only give two examples in English, to give an intuitively appealing impression of the DRS notation. All other examples are from authentic application of the system in German.

[3] In other situations, the more general solution supplied by KALEX might be desirable, i.e. the general purpose orientation of the natural language analyser justifies the supplied solution.

3a *Wenn ein Auto gegen eine Leitplanke prallt, kollidiert es.*

```
1  (for all u1,u2,t1,e1,w1,l1
2      'Auto'(u1) &
3      'Leitplanke'(u2) &
4      prallen(u1,u2,e1) &
5      weg(u1,w1,e1) &
6      loc(u1,l1,e1) &
7      gegen(l1,u2))
8   -> (exists t2,e2
9           kollidieren(u1,e2)) |
10     (exists u3
11          kollidieren2(u1,u3,e2))).
```

In this case, the result provided by the general purpose natural language analyser is evidently too general for the purpose intended by the user. The first alternative in the conclusion denotes a collective interpretation of ***kollidieren (to collide)***, whereas the distributive interpretation suggests itself, regarding the occurence of the distinct colliding objects ***Auto = car, Leitplanke = road delimiter.***

This example furthermore shows that concrete instances of rules, as may be evoked by presenting cases, may need generalization [Wette88].

3b

```
1  (for all u1,u2,t1,e1,w1,l1
2      physisch(u1) & 'Objekt'(u1) &
3      physisch(u2) & 'Objekt'(u2) &
4      prallen(u1,u2,e1) &
5      weg(u1,w1,e1) &
6      loc(u1,l1,e1) &
7      gegen(l1,u2))
8   ->
9      kollidieren2(u1,u2,e1).
```

The fourth example is transfomed into DRS although blind alley is not in the dictionary.

4a *Every street which ends is a* **blind alley**

```
1  (for all e1,u1,u2,u3,u4,t1
2      blind alley(u1) &
3      street(u2,u3,u4) &
4      end(e1) &
5      time_of(t1,e1)          ->     u2 = u1
```

The fourth example DRS is semantically or pragmatically deficient in the sense that the to be defined term (blind alley) is part of the premise. Additionally the temporal interpretation of **to end** and **blind alley** is not correct. All this can directly be changed into the presented DRS 4b.

4b

```
1  (for all e,x,y,z,t
2      street(x,y,z) &
3      end(e,x) &
4      time_of(t,e)     -> blind alley(x)
```

While so far the examples tend to indicate that the generality of the natural language analyser brings forth more than needed for the formulation of rules, the following example reveals the deficiency that in certain cases no accepted German sentence can be found which connects both premise and conclusion of a rule. A possible way out is to enter either premise or conclusion in German (5a) and to add the other by manually editing (5b).

5a *Ein Mensch kollidiert mit einer Leitplanke und ein Schaden entsteht.*

```
 1  (for all u1,u2,e1,u3,u4,e2,u4,u5,u6,u7,u8
 2     'Mensch'(u1) &
 3     'Leitplanke'(u2) &
 4     kollidieren2(u1,u2,e1) &
 5     tim_of(t1,e1) &
 6     t1 = now &
 7     'Schaden'(u3,u4,u5) &
 8     entstehen(u3,u6,u7,u8,e2) &
 9     tim_of(t2,e2) &
10     t2 = now .
```

5b

```
 1  (for all u1,u2,e1,u3,u4,e2,u4,u5,u6
 2     'Mensch'(u1) &
 3     physisch(u2) & 'Objekt'(u2) &
 4     kollidieren2(u1,u2,e1) &
 5     'Schaden'(u3,u4,u2) &
 6     entstehen(u3,u2,u5,u6,e2))
 7  -> 'Ursache'(e1,e2) .
```

Pre-parser While attempting to rearrange or modify the suggested DRS, the user must arrive at a formally correct version in the end. This is supported by a pre-parser, which analyses user modifications and gives specific messages and/or example patterns, when the user attempts to end editing with syntactically wrong expressions. We subsequently present certain typical error situation occuring during editing DRS's suggested by the natural language analyser.

When cancelling sub-expression such as the *(exists t2,e2 kollidieren(u1,e2)* | *(exists u3*) (cf. example 3a, lines 8 - 10) balancing parenthesis over the whole expression, e.g. cancelling a closing parenthesis after the second disjunct is often forgotten. The pre-parser in this case reminds the user by the message:

> ***Mismatch of parentheses***
> ***Any previous error in the whole predicate can cause this message.***

Experience with KALEX revealed that, when changing non-uniform sub-formulae (cf. lines 5 - 10 of example 5a into lines 5 - 7 of example 5b), a number of errors may occur at a time. Lets assume that the user attempts to cancel lines 5,6, and 9 of 5a.

```
 6     entstehen(u3,u2,u5,u6,e2)) &
 7  -> Ursache(e1.e2) .
```

In this case, he/she receives information about the number of detected errors and may then loop through the error situations in the sequence as they are detected by the pre-parser. In every step the pre-parser marks by colouring, where the error was detected. When the user has made some correction, the pre-parser re-examines the whole expression to take into account side effects of a local change.

In the presented example, it would first request *Ursache* to be placed in quotes (*'Ursache'*)[4]. Then it would detect that infix notation does not allow two operators to follow each other (*& ->*). Finally it would warn to use a half stop instead of a period for the arguments of ***Ursache***, i.e. *(e1,e2)*

---

[4] The underlying Prolog interpreter would take the uppercase Ursache as variable, whereas any uppercase or Umlaut string in quotes is transformed into a predicate.

No variable in DRS or missing quotes
E.g. false: Wort false: früh correct: 'Wort' correct: 'früh'

No second operator after '&'

Warning: reference markers are better separated by comma
You can use PF22 to file and reconsult.

Support of Temporary Versions In some case the expert knows different natural language formulations for essentially the same rule (cf. example 3a).

Prallt ein Auto gegen eine Leitplanke, dann kollidiert es.
Ein Auto, das gegen eine Leitplanke prallt, kollidiert.

He/she may not be aware in advance, which of these are rejected or accepted syntactically and which of the latter comes closest to his/her intended rule contents. Hence the system temporarily allows up to 10 concurring versions of the same rule, which may be interactively commented by the user to reduce his/her memory load in the process of filtering out the best approximation. Furthermore, rejected and accepted attempts are marked by different colours to allow fast screening.

Interactive Test of Intermediate States of a Knowledge Element The user may have a hypothesis, which additional cases or which additional problems w.r.t. some (classes of) cases may be solvable by means of the actually conducted knowledge base amendment.

He/she can interactively test such hypothesis by communicating some case facts in natural language (or load them from his/her collection of cases) and ask questions about the case facts, which should (or should not) be answerable by the theorem prover when using the actual knowledge base. For this purpose he/she can select among the alternatives he/she has provided for some intended rule, compare his/her expected results with those provided by the theorem prover, take notes to filter out inadequate solutions, and in the end decide, which version to make permanent.

In the case study around our examples 2a, 3b, 4b (which contains a number of additional rules), we took the following authentical case description (slightly paraphased to be accepted by the natural language analyser).

Am 7.8.1979 fuhr der Angeklagte mit dem PKW nach Hause. Er fuhr auf der Bundesstraße und hatte Vorfahrt. Ein Verkehrsteilnehmer bog aus der S-Straße in die Bundesstraße ein. Der Angeklagte wich dem Verkehrsteilnehmer aus und prallte gegen die Leitplanken, an denen ein Schaden von 2000 DM entstand.

The following legally relevant questions could not be answered *before* augmenting the knowledge base and one after the other could be answered *after* having added the adequate versions of the respective new knowledge elements.

Was war Straßenverkehr?
daß der Verkehrsteilnehmer einbog.
Kollidierte der Angeklagte?
Ja.
Was war plötzlich?
daß der Angeklagte gegen eine Leitplanke prallte.
Gab es einen Schaden?
Ja.

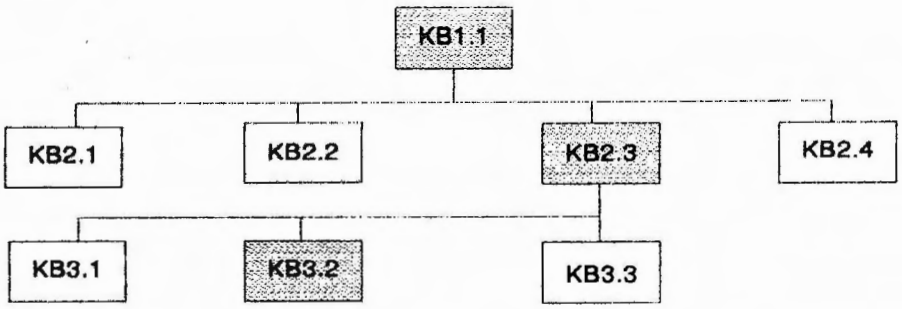
Transparency of Knowledge Elements Whenever the user attempts to enter a new item of knowledge, he/she receives information, whether the involved terms have already been de-

fined and in what knowledge element the definition can be found. Furthermore, the user can query at any time:

- the contents of a knowledge element
- all knowledge elements containing information about some term

Structuring Since in the course of time knowledge about different facets of a domain or even about different domains having some minor part of knowledge in common is to be managed in one system, transparency must also be achieved on a higher level by providing and adequately presenting to the user, how knowledge elements are related to each other. To illustrate this, we now introduce terms which have already been used informally. A ***knowledge base*** is a (ideally self-contained) body of knowledge covering a domain. It consists of a tree of ***knowledge elements***, where the meaning of an arc of the tree resembles [Wachs87]. Each knowledge element may at one time appear in some place of a tree, at another time in a different place and/or tree, but certain restrictions concerning common availability of neighbouring knowledge elements should be adhered to.

Intuitively, one proceeds from general to specific knowledge when traversing a tree from root to leaves. As highly specific knowledge is hardly applicable without some of the general knowledge, of which it is an elaboration, KALEX extends the operations to load and unload knowledge elements to a meaningful path containing the knowledge elements the user has selected. This path is graphically shown to the user, and he can override the system recommendation.



When loading the specific knowledge element KB3.2 the system suggests to include the more general but related KB1.1 and KB2.3

Additional Facilities Concerning unknown grammatical constructions in an intended rule, the user has available a collection of accepted and unaccepted sample patterns from which he/she can infer, which variation of the rule might be accepted. Furthermore, the experienced user can read from the several parts of the linguistic knowledge base, which use(s) of the words he needs are supported by the natural language analyser and what synonyms of the unknown word are available.

Status and Actual Use of KALEX

The system has been implemented in VM/PROLOG on an IBM 3083 mainframe. Additional functions are under work.

The present implementation has been successfully used on the following task:

Given a legal knowledge base (hit-and-run-driving) operational for some variants of **personal injury** as the damaging event of an accident, the system enabled the user to incrementally enhance and interactively test the knowledge base towards **material damage.**

Although quite a number of attempts had to be made to arrive at accepted syntactic formulations leading to acceptable approximations of the intended rules, the estimated time consumption as compared to manually designing DRS's could be reduced to about one half.

Discussion

Facilities and Limitations of the System

Having in mind the considerations of [Tuhri88] about allowing the domain expert to construct, test, and maintain a knowledge base, we have presented the actual status of a system that allows the expert to use the German language for knowledge base construction and to request different views and application scenarios for testing and maintenance. We now examine, to what extent the linguistic problems introduced in the beginning of this text could be solved and what further needs of the domain expert could already be fulfilled.

Syntax KALEX contributes to one aspect of closing the syntactic gap by accepting unknown words. This partly solves the problem of allowing domain specific words. This capacity is so far limited to nouns, if these occur in certain places in a sentence. It would be desirable to have more unknown words accepted and especially to allow unknown verbs and unknown adjectives. The latter needs, however, major amendments of the natural language analyser.

Closing the syntactic gap concerning unknown grammatical constructions is charged to the user, with off line support by sets of patterns of accepted and unaccepted sentences. The usefullness of such patterns has been demonstrated in psychological investigations ([Schma89]) and also turned out helpful in the case study conducted with KALEX, though integration of the collection of patterns would be welcome.

Semantics Since final control of the semantics has to be performed w.r.t. the formal language, it may be asked, whether natural language is of any help at all.

There is psychological evidence again ([Schma89]) that varying an approximation of a formal expression provided by the system on the basis of the user's natural language formulation is easier than newly producing a formal expression. Our case study revealed that this is the more so the more one deals with conceptually complex relations such as temporal or local ones (cf. example 3). These are produced automatically by the natural language analyser by application of a theory about meanings of temporal or spatial expressions. Such theories are of course applied by the human, too, but hard to explicate. (cf. [Hayes85]).

Additionally, the user is efficiently supported in learning which formulation to use for what purpose by always seeing German sentence and resulting DRS in neighbouring windows and being invited to comment in the third window.

In other words: in the actual implementation, based on a general purpose natural language analyser, the user is charged with controlling the formal semantics, but is effectively supported in this task.

Pragmatics The presented knowledge acquisition system interprets all German sentences that it can analyse as object level information, although knowledge representation and theorem prover are capable of dealing with certain kinds of meta-information. This confirms the introductory statement that automated solutions to distinguish between different pragmatics cannot be expected in the near future when aiming at a general purpose natural language analyser. Hence, one either has to make provisions in the environment where the system is used in order to filter the managable object level from other information or one has to reduce ones ambitions from a general purpose system. Here we start with some observations and results, how to proceed concerning the environment.

Transparency and Control For maintenance of knowledge bases, transparency and control are key issues. On all levels, i.e. term, rule, and knowledge element KALEX provides graph-

ical and/or textual informations in user adjustable resolution. This guides the user in tasks such as where to locate a new item or which sub-knowledge bases to use together or alternatively. There are system recommendations, but hardly any rigid constraints to rearrange or reallocate, since for principle reasons (cf. [Wachs87]) we do not aim at warranting a globally consistent knowledge base. Instead, we support competing views expressed in different knowledge elements and the flexible change between views.

Test and Maintenance Knowledge items can be conceived as self-contained expressions having to reflect an intended meaning or can be needed for a certain purpose such as contributing to a required inference.

Most of the discussion of syntax, semantics, and pragmatics so far deals with self-contained details. It has, however, been observed that knowledge items or fragments of knowledge items can be evoked by presenting cases to the expert (cf. [Becke88; Wette88]). In turn it would be desirable to interactively test, whether an evoked (and subsequently syntactically and semantically elaborated) knowledge item is really sufficient to enable a proof and whether there are no side effects of having the new item in the knowledge base. This is not self-evident in our case of a high declarative language such as 1st order predicate logic, since it has been shown that on this level the human cannot mentally control the concequences of newly added axioms (cf. [Evans82]). The interactive test facilities satisfy both needs,

- to test whether the desired effect is achieved when dealing with the case that evoked an item to be added
- to control on other cases whether there are no undesired side effects.

If the knowledge does not prove adequate for either reason, one can either locally adjust a rule to satisfy both needs now, or one can come to the conclusion that the new item reflects a new and non-compatible view that has to be expressed in a competing knowledge element.

Hence the system components illustrating the knowledge base structure and those allowing immediate test of transient states of the knowledge base and the easy navigation between both supports the user in gradually augmenting the knowledge base and at the same time maintain the pre-existing state operational.

Comparison with other Approaches

Natural Language Aspects When compared to new developments in automated natural language based knowledge acquisition such as [Szpak88], KALEX appears less elaborate in the domain model but broader in linguistic coverage of different domains. Furthermore, dialogue aspects and active creation and test of knowledge bases direct our investigation, in contrast to the above text linguistic approach.

This has different consequences. With a knowledge representation language and natural language analyser devoted to the domain of software manuals, [Szpak88] reaches a broader and more discriminating result concerning this preselected domain. This includes a greater variety in detecting different pragmatics. Sentences in a software manual can be interpreted as declarative (describing a property of the software) or procedural (telling how to use the software).

There is no indication, whether the results are transferable to other domains. Since the transfer process is a slight variation in the favourable case, but may turn to be a complete reformulation, it may be argued that the expenditure for covering a new domain may be high.

Nevertheless, a restricted purpose system as the one of [Szpak88] may be the only means in the near future to achieve acceptable pragmatic discrimination. Hence it is also one perspective of our approach to conceive one or several specializations of the so far general purpose system. This will hopefully be easier in our case, since the general purpose system does supply a stable background, which can be enhanced in several different ways.

The Knowledge Assimilation Paradigm [Lefko87] have recently claimed that knowledge acquisition tools should support the user in adequately placing new items of knowledge in a pre-existing knowledge base and, if necessary, to modify (assimilate) a knowledge base automatically. They present a frame based system, where these tasks are performed by subsystem K^n_{AI}. Based on numerial criteria of degree of match, applied to frame structure, they perform the allocation task. Some heuristic rules guide the assimilation process.

This kind of support is highly desirable, when facing growing knowledge bases. It can, however, not be supplied in the above form, when going beyond propositional logic expressiveness or when allowing competing views, as in our case.

That is why KALEX only supports the allocation task. It allows the user to formulate a detail, and have the system present related details. Then the user decides where to allocate the new item and if to modify any old or the new item or to accept competing details.

The Explanation Based Paradigm In contrast to the explanation based learning paradigm of machine learning, the term is also used by [Silve87] for a strategy to base knowledge acquisition on expert explanations (KBAM). As part of this approach, utterances are attempted to be classified as *foundational* (i.e. declarative), *constructive* (i.e. procedural), or *conjoint* (i.e. organisational and controlling). Furthermore, overall structural properties of explanations are taken into account, e.g. degree of indetermination (e.g. procedural use of not yet declared terms).

This means that a number of important pragmatic distinctions are paid attention to and that the system tolerates underdetermined transient states and keeps track for the user, where he has to supply additional information.

Some of these results would be desirable for KALEX. One of them certainly is the distinction between pragmatic categories. Another is the being reminded of undefined terms, which we would need whenever the natural language analyser has accepted a word although there is no dictionary entry.

On the other hand, KBAM needs pre-parsing, i.e. operates on a conceptual representation of the explanation supplied by NLP ([Birnb79]). I.e. the coverage of KBAM is limited by the coverage of NLP, which also applies to any attempt to extend KBAM to new domains.

The Sloppy Modelling Paradigm

> *"In the sloppy modeling paradigm, the goal is a modeling style where user and the system's structuring and learning routines are regarded as independent partners that both work towards having a well-structured and complete model"([Wrobe87]).*

Sloppy modelling systems as suggested by [Wrobe87] and partially implemented in BLIP are characterized by a number of properties, most of which can also be found in our system.

Flexibility in the sense of allowing the user the largest possible number of state transitions instead of rigid menu guidance is present in the KALEX-System (as well as in BLIP). All knowledge base entries can be withdrawn at any time, i.e. operations on the knowledge base are *reversible*. This goes beyond immediate file management operations but includes *reversible* considerations about what knowledge elements or sub-knowledge bases are involved by a user's attempt to make a local change. These are reported to the user, who then has the chance to confirm or reject the extension of the operation.

The system indicates what parts of the knowledge base are influenced by a local change (*liveness*). Major advantages of BLIP concern automated quality checks (*integrity*) and machine learning elements (*transitionality*), both of which have not been aimed at in KALEX. On the other hand, KALEX is more helpful than BLIP in allowing *inspection* of smaller or larger sub-knowledge bases upon user's request.

Conclusion

KALEX is a natural language based knowledge acquisition system integrated into a natural language oriented knowledge based system architecture. It accepts German rule sentences and displays them together with their automatically generated logic DRS notation. This allows th user to gradually learn the semantics applied by the system to syntactic classes of sentences. Furthermore, can he/she factually exercise control over the knowledge base by having parallel versions available for editing, selectively loading ect.. This process is mentally supported by giving him/her additional space in the screen for his/her personal comments concerning the history, validity and usefulness of versions. Furthermore, the risk of leaving behind a non operational knowledge base is reduced by making transparent, which knowledge are affected by an intended local change.

KALEX's major strength is a comparatively broad coverage of a robust general purpose natural language analyser and several functions to provide transparency and testability of knowledge elements and bases.

Further work has to be done to increase the semantic quality and the pragmatic discrimination. It is assumed that general solutions to both tasks are not at hand. That is why a collection of restricted purpose systems, each devoted to a small or medium class of applications, is advocated for. These could all start out from what we have, but would be individually and specifically enhanced towards semantics and pragmatics of their respective domains.

References

[Barne86] Barnett, B., Lehmann, Hu., and Zoeppritz, M., *A Word Database for Natural Language Processing,* 11th International Conference on Computational Linguistics, Proceedings, Bonn, Germany (August 1986), pp. 435-440.

[Blase88] Blaser, A. (ed). *Natural Language at the Computer. Contributions to Syntax and Semantics for Text Processing and Man-Machine-Communication* Springer, Berlin Heidelberg New York (1988)

[Becke88] Becker, B. *Towards a case-oriented concept for knowledge acquisition* Proceedings of the European Knowledge Acquisition Workshop (EKAW'88) Bonn, (Juni 1988)

[Birnb79] Birnbaum, L. and Selfridge, M. *Problems in Conceptual Analysis of Natural Language,* Research Report #168, Yale University (1979)

[Breuk87] Breuker, J., Wielinga, B., van Someren, M., de Hoog, R., Schreiber, G., de Greef, Bredeweg, J., Billault, J.-P., Davoodi, M., and Hayward, S. *Model-Driven Knowledge Acquisition: Interpretation Models, Deliverable Task A1* Esprit Project 1098, Memo 87, (1987)

[Diede87] Diederich,J., Ruhmann,I., and May,M. *KRITON: A knowledge-acquisition tool for expert systems* Int. J. Man-Machine Studies (1987) 26, 29 - 40

[Evans82] Evans,J. St.B. T. *The Psychology of Deductive Reasoning.* Routledge and Kegan Paul, London (1982)

[Grosz87] Grosz, B.J., Appelt, D.E., Martin, P.A., and Pereira, F.C.N. *TEAM: An Experiment in the Design of Transportable Natural-Language Interfaces* Artificial Intelligence **32** (1987), 173-243

[Guent86] Günthner, F., Lehmann, Hu., and Schönfeld, W. *A Theory of Representing Knowledge* IBM Journal of Research and Development, Vol. 30, No. 1. 39-56 (1986)

[Haft87] Haft,F., Jones,R.P., and Wetter,Th. *A Natural Language Based Legal Expert System Project for Consultation and Tutoring - The LEX Project* Baltimore (1987) ACM

[Hayes85] Hayes, P.J. *The Second Naive Physics Manifesto* in: J.R.Hobbs and R.C.Moore(eds.): Formal Theories of the Commonsense World; Ablex, Norwood NJ (1985)

[Kamp81] Kamp, H., *A Theory of Truth and Semantic Representation* Groenendijk et al. (eds.), Formal Methods in the Study of Language. Mathematical Centre Tract, Amsterdam (1981). Reprint in: Groenendijk et al. (eds.), Truth, Representation and Information. GRASS 2, Dordrecht: Foris

[Lefko87] Lefkowitz, L.S., and Lesser, V.L. *Knowledge Acquisition as Knowledge Assimilation* 2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada (October 1987) (modified version in Int. J. Man-Machine Studies (1988) 29, 215-226

[Lehma85] Lehmann, Hu., Ott, N., and Zoeppritz, M. *A Multilingual Interface to Databases* IEEE Database Engineering Bulletin 8.3, (1985)

[Nisbe77] Nisbett, R.E. Wilson, T.D. *Telling more than we can know: Verbal reports on mental processes* Psychological Review **84**, 231-259 (1977)

[Schma89] Schmalhofer, F. and Kühn,O. *The Psychological Process of Constructing a Mental Model when Learning by Being Told, Learning from Examples, and Learning by Exploration.* Ackermann, D. and Tauber, M. (eds) Proceedings of the 8th Interdisciplinary Workshop on Informatics and Psychology - Mental Models and Human Computer Interaction, Schärding 16.5 - 19.5 (1989) (to appear Elsevier, North-Holland Amsterdam)

[Schmi89] Schmidt, G.R. *Synoptische Assimilation fachlichen und sprachlichen Wissens für natürlich-sprachlich orientierte Expertensysteme - Methoden und Prolog-Implementierung* Diplomarbeit an der Fachhochschule Heilbronn und Universität Heidelberg im Fachbereich Medizinische Informatik (to appear in July 1989)

[Schoe85] Schönfeld, W. *Prolog Extensions Based on Tableau Calculus* Proceedings of the 9th International Joint Conference on Artificial Intelligence, Vol. 2, Los Altos, USA, 730-732 (1985)

[Silve87] Silvestro, K. *An explanation based approach to knowledge base acquisition* 2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada (October 1987) (modified version in Int. J. Man-Machine Studies (1988) 29, 159-169

[Szpak88] Szpakowicz ST. *Semi-Automatic Acquisition of Conceptual Structure from Technical Texts* Proceedings of the 3rd Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada (November 1988)

[Tuhri88] Tuhrim, St., Reggia, J., and Floor, M. *Expert System Development: Letting the Domain Specialist Directly Author Knowledge Bases* Hendler, J. (ed) Expert Systems: The User Interface, Ablex Publishing Corporation (1988)

[Wachs87] Wachsmuth, I. *On Structuring Domain-Specific Knowledge* Lilog-Report 12, IBM Deutschland, (March 1987)

[Wette88] Wetter, Th., and Schmalhofer, F. *Knowledge acquisition from text-based think-aloud protocols: specification for a legal expert system* Proceedings of the European Knowledge Acquisition Workshop (EKAW'88) Bonn, (Juni 1988)

[Wrobe87] Wrobel, S. *Design Goals for Sloppy Modeling Systems* 2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada (October 1987) (modified version in Int. J. Man-Machine Studies (1988) 29, 461-477

[Zoepp84] Zoeppritz, M. *Syntax for German in the User Specialty Languages System* Niemeyer, Tübingen (1984)

Acknowledgement We want to thank Hubert Lehmann and Magdalena Zoeppritz, who made this investigation possible by their permanent contributions to the natural language analyser and who encouraged and supported us to attempt the presented approach. KALEX has been designed within the scope of a thesis. We want to thank Prof. Leven, who provided the opportunity to build upon this theme. Ralf Nüse provided some experimental results about availability and syntax of rules. We also want to thank him. Last not least we want to thank Michael Schmitz, who specified and implemented the structuring component of KALEX.

Appendix1

Diebstahl ist die **Wegnahme fremder beweglicher Sachen** in der Absicht, sich dieselben **zuzueignen. Sache** ist klar, braucht man nicht näher zu definieren. Eine **fremde Sache** ist eine Sache, die jemanden anderem gehört. **Beweglich** ist auch klar, braucht man nicht näher zu definieren. Die **Wegnahme** einer Sache ist der Bruch fremden **Gewahrsams** und die Begründung eigenen Gewahrsams. **Gewahrsam** ist ...

A Model for the Knowledge Analysis Process

Steven Twine

Key Centre for Software Technology,
Department of Computer Science,
University of Queensland,
St Lucia, Q 4067.
Australia.

Abstract.

Knowledge analysis is the process of acquiring, interpreting and representing knowledge about some Universe of Discourse and about some specific problem-solving process operating on the objects in that Universe of Discourse.

In order to study (and possibly integrate) the various methods (sometimes called methodologies) that have been proposed for performing this activity, we will require a specification language to describe the components of a method. This language will be used to present a high-level abstract model of the knowledge analysis process.

We can refine this abstract model to arrive (eventually) at the specifications for individual methods. The major distinctions between different classes of methods (such as the role of an intermediate representation or the separation of domain conceptualization and problem-solving task) can be clearly seen in this model. Thus, it provides a rational taxonomy for the analysis of knowledge analysis methods.

Finally, I will indicate how this detailed model can indicate the possibilities for integrating different knowledge analysis methods, and integrating knowledge analysis methods with existing "mainstream" software development techniques. This model can also be used to delimit the scope for automatic support of the knowledge analysis process (using CASE technology or "expert systems for knowledge analysis").

---

1. Introduction.

Knowledge analysis is the process of acquiring, interpreting and representing knowledge about some Universe of Discourse and about some specific problem-solving process which operates on the objects in that Universe of Discourse.

Wielinga [17] has suggested that "building knowledge-based systems is largely a modelling activity that can (and should) be supported by modelling languages and template models." The basic idea behind the KADS project (as stated in [17]) is to "view the process of KBS development as a *modelling* [my emphasis] activity." This statement neatly reflects a (now) widely-accepted view in the knowledge engineering community. The trouble begins, however, when we begin to discuss what we mean by a "model".

There seem to be two (closely related) models required in a knowledge-based system. The unelaborated term "model" is often used (and confused) to refer to either of these two models or to their integration. The process of constructing a knowledge-based system involves constructing an *integrated* specification of these two models. In this sense, knowledge analysis is, indeed, primarily a modelling process.

The first of these two models is a representation of the expert's knowledge about some Universe of Discourse (a *UoD* model). Typically, this UoD model describes the permitted entities in the UoD and the permitted relationships between these entities. The UoD model acts as a grammar for the sentences that may be communicated (recorded or inferred) about the Universe of Discourse. It typically includes both intensional aspects (the grammar) and extensional aspects (sentences conforming to the grammar). In terms of Ontological Analysis [3], this UoD model is the *static* ontology. In the terminology of Addis [1], the UoD model contains all (and only) relevant asserted and inferential knowledge.

The second of the two is a model of some specific problem-solving process to be performed in that Universe of Discourse (a *task* model). The task model describes some problem-solving process which operates on the entities and relationships specified in the UoD model. In terms of Ontological Analysis, the task model contains both the *dynamic* ontology and the *epistemic* ontology. Addis would call this heuristic knowledge.

Furthermore, these two models may exist in many different states. Wielinga [17] talks about *two* (quite different) models: an *analysis* model which is "a model of the real world domain" and a *design* model which is "a model of the computational artifact." Of course, these model classifications are completely orthogonal to the previous model classifications: we could quite realistically require a UoD analysis model, a task analysis model, a UoD design model and a task design model.

We clearly have many different models or representations which are required during the knowledge analysis process. This raises obvious questions. How are these models related? How do we construct a model? How does one model (already developed) *guide* us in the development of subsequent models (e.g. how does the analysis model contain, or even determine, the design model)?

In this paper, I will construct a model of the knowledge analysis process that provides a basis for answering such questions as these.

We can regard any modelling process as a specialization of a general *transformation* process where the object that is transformed is an evolving representation of the UoD and task models. At the highest level of abstraction, we can, therefore, regard a knowledge analysis method as a process which transforms input information into output information. Any complete specification of such a process must include a specification of the permitted structure of the input information and the output information, as well as a specification of the transformation.

There is a clear relationship between a knowledge analysis *method* and the knowledge analysis *process:*

```
process ≡ method(situation)
```

In other words, a specific knowledge analysis *process* is the *application* of some (implicit or explicit) knowledge analysis *method* to a specific situation, yielding a specific result.

This way of looking at knowledge analysis (which I call the *process* view) leads naturally to a very abstract initial model of the knowledge analysis process. This initial model is *minimal* insofar as it specifies the minimum number of different *types of facts* that can be known about the components of this process.

Abstraction is an intellectual operation which selects a *subset* of the available facts about an object as being relevant *for a given purpose*, and disregards (or suppresses) any other facts as being irrelevant (for that purpose). If you known N facts about an entity, but you select only N-P facts as of interest, then you have *abstracted* from those remaining facts. Conversely, if you have N facts, of which P are in the "scope of abstraction" and then you add an additional M facts to that scope, you have *de-abstracted* from P facts to $P+M$ facts.

The "scope of abstraction" of this process model is gradually widened by explicitly specifying additional types of facts about the elements of the process model (processes, subprocesses, their inputs, outputs and definitions) to that initial model. In other words, the process model is developed further by a process of *rational de-abstraction* from the initial model, which carefully selects a *single* additional type of fact to add to the model at each step (and which carefully justifies that extension). If we continued this process for long enough, we could (theoretically) arrive at a specification of any specific knowledge analysis method.

Conversely, we can *classify* the methodological elements of existing methods using this process model: it provides a taxonomy (or classification scheme) over models which induces a corresponding taxonomy over the (analysis and design) processes which use and produce these models. These taxonomies are rational, in the sense that they are derived from *deep properties* of the knowledge analysis process and not from some *ad hoc* collection of features.

2. The Process View of Knowledge Analysis.

If we view knowledge analysis as a transformation process, then this implies that we should be able to use some classical process description techniques to construct a model of the knowledge analysis process. This is true to a certain extent, but classical process description techniques may not be adequate for describing the entire knowledge analysis process. It is important to realize that these limits of description do not reflect a limit in the process view of knowledge analysis, but an inherent limit in the existing process description techniques (because they were designed for specifying purely formal object-level processes).

The idea of considering the evolving models as metaknowledge (in this case, a meta UoD model) is not a new one. However, we can easily extend this idea to consider the knowledge analysis *process* as a meta task model: a problem-solving process over the metaknowledge (model components) in that meta UoD model.

What is a process description? Clearly, at the coarsest level of description, a process consists of three main components:

1. some input information
2. some output information
3. a transformation from input to output

Now, recall that a design process is the application of a method (or design procedure) to a specific situation. Hence, we can generalize from the specific instantiation of the process to a general description of the process, which is independent of the specific input and output:

1. a specification of the possible input types and their associated constraints (i.e. any permitted input must be an instance of this type)
2. a specification of the possible output types and their associated constraints (i.e. any possible output must be an instance of this type)

The input and output instances constitute (different states of) the evolving representations of the UoD model and task model. These are *passive* components: they represent *knowledge* that can be operated upon. Therefore, the input and output *types* define the *prescriptive grammars* of the languages used to express these representations.

3. a specification of the transformation that converts instances of the input types into instances of the output types

The various processes represent transformations of that knowledge (i.e. those model specifications) from one state to another. These are *active* components: they represent *operations upon knowledge* (i.e. analysis and design decisions taken).

If we could regard the process as an executable program, this description is the program code (Sintzoff's *meta-program* [18]), while the process is a specific "run" of the program (which ultimately generates a program!).

If we wish to give a complete description of the (static) interactions between a set of processes and models, then we must also describe:

4 the *knowledge dependencies* between design processes (which *constrain* the sequence in which the specified design processes must occur).

More precisely, we define knowledge dependencies between a process and the model representation (input/output types) that it uses/produces. The indirect knowledge dependencies between processes can, therefore, be derived:

```
IF process X produces model representation M
   AND process Y uses model representation M
THEN process Y is knowledge-dependent on process X
```

Finally, in order to execute these processes (and actually perform knowledge analysis), we must describe:

5. the *control dependencies* between processes (which *determine* the sequence in which processes actually occur)

The knowledge dependencies do not directly correspond to control dependencies between processes. However, they do indicate which knowledge is shared between processes, and therefore, induce a weak partial order over the execution of these processes. In this way, the knowledge dependencies represent a weak circumscription of the possible set of control dependencies over processes. The control dependencies can be more precisely specified with *explicit preconditions*.

How can we describe a transformation process? We can describe such a process in a number of different levels of detail. We could describe it *operationally* (by providing a processor that performs the process), or *declaratively* (by providing formal specifications for the input types, output types and transformation functions).

At this point, we must make an observation that will be explored in more detail later in this paper: not all input types can be described by formal languages, therefore not all processes can be described by formal languages. This leads to two conclusions:

1. We cannot provide a full operational semantics for such a process (because some parts of the process are inherently not automatable). In this case, the operational semantics would include a (probably unpredictable) human processor! This is rather like defining an operational semantics of human expertise by pointing at the expert.
2. We cannot provide a full declarative semantics for such a process (because some parts of the process cannot be specified as a formal transformation).

The solution is to use a number of process description languages, with different levels of detail.

The *signature* of a set of processes identifies (arguably) distinct processes and their input/output knowledge, without specifying the structure of those input/output types, or formally defining the transformation (which may not be possible). The signature could be expressed in a language which is closely related to the data flow diagrams of conventional system analysis.

For those input types and processes which are amenable to formal description, formal knowledge and process description languages can be used. For the other input types and processes, informal descriptions can be given. It is an assumption of this model that all output types can be formally described. The *specification* of a set of design processes provides a precise (but not necessarily formal) specification of the input and output types, knowledge dependencies, control dependencies and the transformation relationship between input instances and output instances.

Finally, the *implementation* of a set of design processes provides the precise (formal) algorithms and (informal) heuristics that are used to perform the transformations (in a way that always satisfies the constraints of the specification) together with a control framework for executing the algorithms. If we restrict this to formal representations and algorithms, we have Sintzoff's metaprogram.

"Knowledge" Flow Diagrams. These knowledge flow diagrams provide a concise graphical notation for representing processes and the knowledge that flows between them, together with their associated (explicit or implicit) knowledge bases (which store that knowledge).

A language for specifying knowledge flow diagrams must contain the following three types of components:

1. active components called *processes*.

These are used to model the various steps in a method. Processes are named, and may be formally defined (where this is possible) using an appropriate process specification language (such as those used in classical software specification) or by natural language (if only informal definition is possible).

2. passive components called *knowledge stores*.

These are used to model the input and output types of those processes. If the knowledge store can be formally defined, then its permitted contents can be specified using some knowledge specification language (such as NIAM [20] [22] or CL [2]). If only informal definition is possible, then the contents of the knowledge store can be syntactically specified (as character strings etc) or semantically described by (informal) natural language text.

3. connectors called *flows*

These flows are used to model the knowledge dependencies between passive components (knowledge stores) and active components (processes).

Figure 1 shows the symbols used to depict these components:

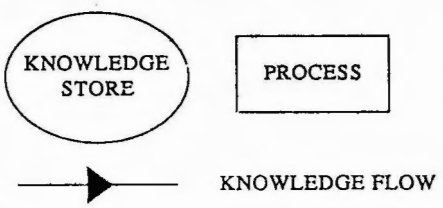


Figure 1. Knowledge Flow Diagrams.

It is important to understand that this is a formal language, even though we use a graphical representation. Many people *mistakenly* assume that all graphical languages are inherently informal.

Obviously, there is a close family resemblance between these knowledge flow diagrams and more traditional data flow diagrams. Both diagrams can describe processes which take input information and produce output information. However, there is an additional type of component available in most traditional data flow diagram languages which is not available in knowledge flow diagrams: the external entity or environment.

The external entity is a source of information that resides outside the computer information system. It is distinguished from a normal data store because the contents of an external entity cannot be formally specified. Typical external entities are customers or other organizations. In a data flow diagram for knowledge analysis, the expert would probably be represented an external entity, but the other sources of expertise (protocols, example cases and so forth) would be represented as data stores.

In a knowledge flow diagram, no distinction is made between "internal" knowledge stores and "external" knowledge stores. The analysis process transforms knowledge from one state to another. Initially, as we will see, the knowledge will be informally represented (implicit within

data) and it must be transformed into a formally represented (explicitly structured) data structure of some form. The distinction between implicit and explicit knowledge is not made in these top-level knowledge flow diagrams, but will be introduced as we elaborate the initial process model.

Two types of facts can be directly stated in this graphical language:

```
Component is of Component Type (Active or Passive)
Component is knowledge-dependent on Component
```

3. An Abstract Model of the Knowledge Analysis Process.

Let's consider the traditional model of knowledge analysis, as depicted in figure 2. The five steps in this model have been taken directly from the text of "Building Expert Systems".

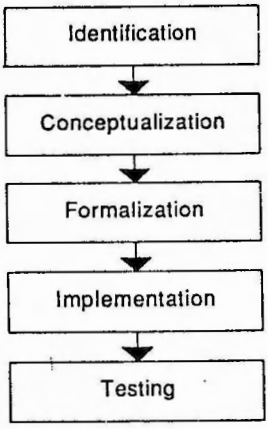


Figure 2. A Traditional Model of Knowledge Analysis.

This model describes the knowledge analysis process as a sequence of five steps: identification, conceptualization, formalization, implementation and testing. In other words, the knowledge analysis process is refined into five subprocesses, which must be performed in sequence. This is the traditional "knowledge analysis life-cycle" that has been accepted by many workers in the area.

However, this model is neither sufficiently precise nor flexible. It does not specify the input or output structures of any step, nor does it specify the transformation performed during each step. Furthermore, this model can only describe design processes that consist of a fixed sequence of steps, without iteration. Finally, the division into subprocesses seems quite arbitrary. There appears to be no simple predicate for dividing a design process up into these five stages.

We need a more flexible and precise model, with a clear foundation and which can be derived, in a straightforward way, from that foundation.

Returning to our process view, we can at the highest level of abstraction, describe the knowledge analysis process as a transformation from an initial (integrated) model to a final (integrated) model. This is shown as a knowledge flow diagram in figure 3.

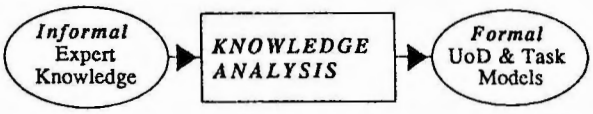


Figure 3: Knowledge Analysis as Transformation.

Now, let us take a closer look at these three components.

The Input. The initial information is typically an unsuitably structured and possibly incomplete mass of information expressed in one or more human languages (typically a combination of unrestricted natural language and some specialized, and possibly formal, language). This information has been extracted from the expert, or from other sources, during a (presupposed) elicitation phase. As a whole, however, we can be certain that this initial information is informal. More precisely, this initial information can be syntactically formalized (eg a set of character strings) without difficulty but this information is highly context dependent, so the meaning (i.e. denotation in the UoD) of a sentence is not readily available. In other words, the initial input is only *uninterpreted data*. As Breuker and Wielinga [5] have aptly phrased it: "these verbal data do not speak for themselves." Consider the following example:

```
Martin has a blood pressure 100/80 and a temperature of
40.8.
```

While we could easily write a data description for this sentence (a series of character string separated by white space and terminated by a "."), the meaning of the sentence is not unambiguously specified. All sorts of questions remain to be answered.

What does the blood pressure rating mean? What scale is it in? Is 100 the systolic rating or the diastolic rating? And what about 80? What scale is the temperature measured in? Is Martin a human patient or a laboratory animal?

All of these questions must be answered before we can extract the underlying (deep structure [9]) UoD facts from this (surface structure) sentence.

Clearly, the most important observation is that this input cannot, *by definition*, be represented in a formal language because the *essential* task of knowledge analysis is the "transformation of a set of [unstructured] data into some abstract [formal] structure structure" [17].

Transformation Procedure. A modelling process is essentially a process of analysis and formalization; the goal of modelling is to arrive at an appropriate formalization (which implies an appropriate interpretation) of the data.

Sowa [19] attributes to Perlis the remark that it is "not possible to map informal specifications onto formal specifications by means of a formal algorithm." This is true, simply because an informal specification leaves most of the interpretation conventions implicit. Therefore, this transformation of expressed knowledge from an informal representation to a formal representation requires (at least) some existing knowledge of the relevant UoD.

It is (at best) possible to provide informal, or heuristic, guidance to map an informal specification onto a formal specification. These informal rules must help the expert and the knowledge engineer explicitly identify the relevant assumptions, and express the initial information in some formal language as guided by these assumptions.

Output. Ultimately, the output of this knowledge analysis process must be a suitably structured, formal representation of that initial knowledge which is expressed in some computer-processable language. For example, we could express the knowledge in our example sentence in a typical frame language as follows:

```
Frame Martin
  Is-a Patient
  Slot systolic-bloodpressure
       Value: 100
       Unit:  mmHg
  Slot diastolic-bloodpressure
       Value: 80
       Unit:  mmHg
  Slot sublingual-temperature
       Value: 40.8
       Unit:  degC
```

This frame language presumably has some formal semantics (probably only an operational semantics in the form of an interpreter, though!).

This output information is clearly both formal and context-free. It is now context-free because the context assumptions have been identified (during the informal modelling process) and added to the evolving model. In other words, this model consists of *interpreted* (within the context) *data*. In fact, it is the presence of this explicit level of (operational or declarative) interpretation that distinguishes data from knowledge.

Input-Output Equivalence. A hidden assumption in this discussion of the transformation function is that it produces a final model which is, in some as yet undefined sense, *equivalent* to the initial information.

Of course, there can be no possibility that the two knowledge stores are *equal*. As Rappaport and Gaines observe in [16]: "there is a loss of information due to the limits of the artificial representation." In other words, the transformation process necessarily involves an abstraction process which suppresses some details in favour of others. However, it is important to recall that this model has a purpose, and we would like to ensure no salient information is lost *relative to that purpose*. In other words, we would like to ensure that the initial model and final model are semantically equivalent. However, it is not possible to give a rigorous (and formally checkable) definition of this equivalence, only an intuitive one.

We could regard the initial information as defining, part by intension (in the form of "domain theory") and part by extension (in the form of example cases and protocols), two sets:

S_U = { s_U | s_U is a possible state of the UoD }
T_U = { (s_{U1}, s_{U2}) | (s_{U1}, s_{U2}) is a possible transition between states }

Similarly, we could regard the final integrated model M as defining, purely by intension, two sets:

S_M = { s_M | s_M is a permitted state of the UoD model }
T_M = { (s_{M1}, s_{M2}) | (s_{M1}, s_{M2}) is a permitted transition between states

From this view, the intention is that the transformation process constructs an integrated model such that the following conditions hold:

There is a 1:1 mapping between S_U and S_M
There is a 1:1 mapping between T_U and T_M

This is essentially the same view of validation that is used by Benbasat and Dhaliwal [4]: "The degree of validity of the knowledge acquisition process is the degree of homomorphism between the representation system and the system that it is supposed to represent."

Of course, because there can be no formal definition of S_U or T_U, it is not possible to demonstrate, within the confines of a purely formal system, that these conditions hold. There can be no formal guarantee of semantic equivalence between initial information and final model. This is precisely *why* the resulting model must be validated.

If the input and output of the transformation were formally specified, then validation would be unnecessary (even if the transformation itself was informal), because semantic equivalence could be proved using purely formal means (ie verification).

This reflects an interesting light on certain issues in mainstream software development. If the input, output and transformation are all formally specified, then there is no need to perform any validation or verification. This can be guaranteed as a logical consequence of the definition of the transformation function. This is the case for work in automatic program transformation. However, if the input and output are formally specified but the transformation is not (as is the case for top-down refinement), then a verification step is required, to demonstrate formally that input and output are semantically equivalent. This is the case for most software development, such as the manual refinement of a specification into an implementation.

4. Distinguishing Specification from Implementation.

the essential problem, therefore, is that the transformation from an informal representation to a formal representation is a grand (and inherently unformalizable!) leap. Such a great leap is difficult to control and to validate. Therefore, it makes an unsuitable model for controlling the knowledge engineering process.

There is also another problem: informal processes cannot be fully automated. This places a severe limit on the application of CASE technology (and expert system technology) to support knowledge analysis tasks. We would like to refine this initial model further, with the aim of reducing the informal component.

To make this refinement, we will introduce the distinction between *conceptual* (= epistemological [5]) representation and *machine* representation. This presumes that the results of an analysis process can be expressed in some (conceptual) representation which is independent of its eventual implementation in computer data structures. In other words, we can define two (partially dependent) classes of model representations: conceptual specifications and implementations. A change in the conceptual specification *implies* a change in the corresponding implementation, but a change in the implementation *does not imply* a change in the corresponding specification.

A conceptual representation is solely concerned with the semantic *specification* of knowledge, and specifically excludes any machine-oriented (implementation or interface) considerations. This corresponds closely to Johnson's mediating representations [12] [13], which must "capture all of the relevant facts, opinions and relationships in the domain [UoD] and exclude nonsense and contradiction." [13] In other words, it is a grammar which prescribes the knowledge that may be recorded in any state of the UoD model, as well as the permitted transitions between those knowledge states (which defines the task model).

The basic idea, then, is to decompose the initial transformation process into two subprocesses, as shown in figure 4.

The first subprocess is still an informal process. However, the second process starts with a formal conceptual model and finishes with a formal computational model. This means that the second process *can* (but need not) be specified as a formal process, and hence can be automated. As an example, this division is exploited in [20] where the (informal) NIAM conceptual schema design procedure is used to construct a conceptual model and a formal algorithm is used to implement this model as a set of KEE frame data structures.



Figure 4. Introducing an Intermediate Conceptual Representation.

The congruences between the specification phase and an informal transformation, and between the implementation phase and a formal transformation, are common and desirable, but not obligatory. For example, KADS includes an informal specification phase and (currently) an informal implementation phase. There are two reasons for this. First, the KADS conceptual model does not contain all of the information about the UoD and task models that is required for their implementation (i.e. the specification is not complete). In other words, the second (implementation) subprocess must contain some analysis processes (involving informal inputs). Second, KADS currently uses an inherently informal top-down refinement process that moves from an analysis model, through a coarse design model and a detailed design model, before arriving at the final implementation.

It is interesting to observe that many prototyping methodologies rely on *executable* specifications which mix specification and implementation details. Such a "specification" is not free of implementation bias. This often leads to "scaling-up" problems when moving from the prototype to the real system, because these implicitly specified implementation details (contained in the specification) do not fit the requirements of the real system.

Specification Phase Support. Support for the first (informal) subprocess can vary from weak linguistic heuristics to template models. In any case, it must provide an interpretation framework (that is, a set of categories or types). The verbal data must be expressed as instances of the categories provided in this framework.

Linguistic heuristics, such as those provided by Dubois, Hagelstein and Rifaut [6], often suggest the interpretation of data on linguistic grounds. For example,

> Phenomena denoted by nouns in the discourse are more likely to correspond to objects, whereas verbs tend to correspond to associations. However, verbs might also correspond to events, especially when they have circumstancial complements, which then correspond to associations granted to these events. [6]

These heuristics often require expert knowledge of the UoD context to disambiguate difficult cases (e.g. when does a verb correspond to an object and when does it correspond to an association). These heuristics tend to be weak because they are formulated in terms of surface linguistic structures that can be used to reflect a given deep structure in a number of very different ways. Linguistically-based heuristics can be provided for interpreting information about the problem-solving knowledge (eg KRITON's protocol analysis heuristics [14]) as well as information about the UoD.

An alternative heuristic for this interpretation process is the *conceptualization* or *visual-auditory transformation* heuristic suggested by Nijssen [15]:

> Take the example data in your hand. Pretend that you are telephoning your sister in the Himalayas. Read the example over the phone to her.

This heuristic is coupled with a number of immediate quality checks to ensure that the resulting UoD model is well-structured. This heuristic can only be used to interpret information about the UoD model. Note that this heuristic does not rely on surface linguistic features, but extracts the expert's interpretation of the sentence's deep structure (in terms of entities, roles and labels) [20].

An alternative approach, which is stronger but less flexible, is the use of *template models*. One example of the use of template UoD models is the prototype "expert system for information analysis" developed by Falkenberg, van Kempen and Mimpen [8]. This system relies on some initial uninterpreted data, and a knowledge base of template UoD models, to construct a UoD model that correctly interprets the initial data.

This solution presupposes that a "closely related" UoD has been analysed before (to provide the appropriate template model). If this is not the case, then there is no starting point for the analysis and no guidance for the knowledge engineer. Of course, even if there is an appropriate template model, the knowledge engineer will still need specific guidance about which aspects of this template model correctly describe this (slightly different) UoD and which aspects must be changed. If there are a number of appropriate UoDs, then the knowledge engineer needs guidance in selecting the correct alternative and in resolving any conflicts between template models. The prototype described in [8] appears to do this by propagating arbitrarily-assigned probabilities.

Finally, it is not possible to give a formal rule for detecting when two UoDs are "closely related" until we have performed a preliminary analysis (at least) of the UoD (to provide a vocabulary for expressing the decision rule!). In [8], the choice between alternative template models is made by performing an initial interpretation using weak linguistic heuristics.

A similar approach has been suggested for interpreting initial problem-solving information using KADS interpretation models [5]. This approach suffers from similar difficulties in determining when an interpretation model is relevant and how a relevant model must be amended to fit the specific UoD. However, there is at least the possibility of formulating explicit rules to determine when a template model is appropriate, because such rules can be expressed in terms of the (already specified) UoD model.

5. Distinguishing Transformation from Validation.

We can also distinguish transformation (model *generation*) from validation (model *checking*) on the basis of partial interdependency. We can, therefore, revise our model to include the additional fact (not specifiable by distinct graphical symbols) that a process is a validation or a transformation:

```
transformation: old-model → new-model
validation: old-model × new-model → {valid, invalid}
```

This revised model is shown in figure 5.

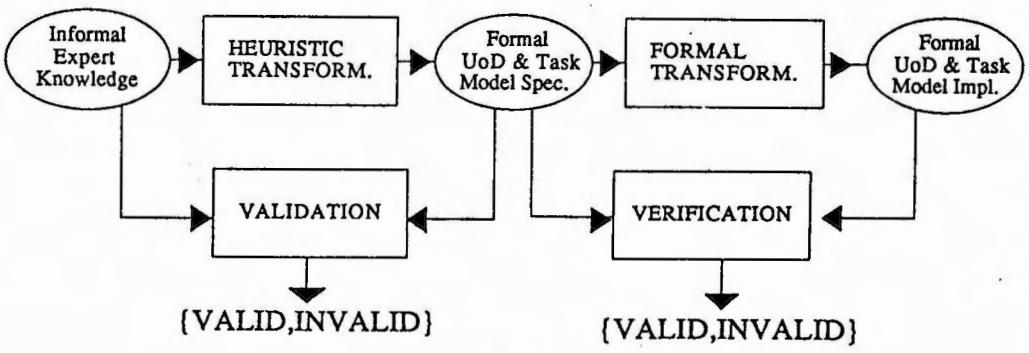


Figure 5. Distinguishing Transformation from Validation.

Effective validation requires that the expert *understands* the implications of the model representation and *agrees* that they correctly represent his or her knowledge. *Direct* validation requires that

the UoD expert must validate the substance of the model representation *per se* whereas *indirect* validation uses a (potentially formal) transformation to derive some significant implications from the model representation and tests the expert's agreement with these implications. If the expert agrees with the implications, we conclude that the 1:1 mapping between UoD and model is preserved.

We can use the distinction between conceptual representation (specification) and machine representation (implementation) together with the distinction between direct and indirect validation to provide a two-dimensional space for plotting validation strategies. This is shown in figure 6.

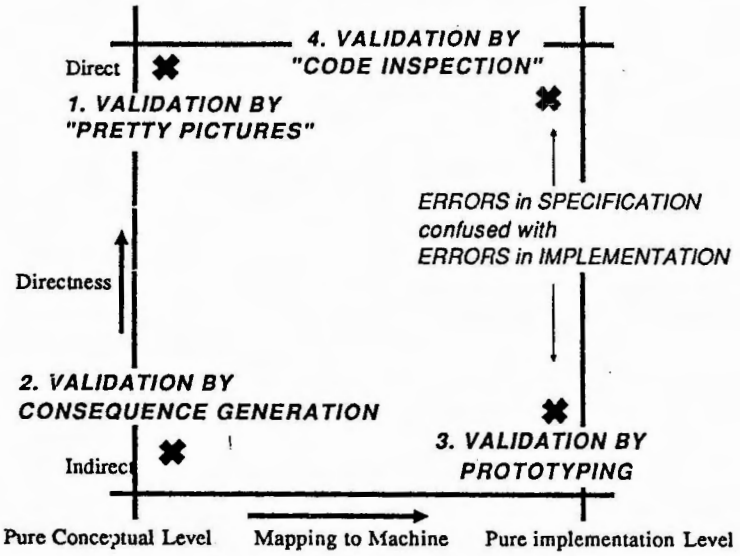


Figure 6. Validation Strategies.

Along one axis, a strategy is placed according to whether the validation exercise uses a conceptual representation or a machine representation. Along the other axis, a strategy is placed according to whether the model's intension is directly validated, or whether some consequences of that intension (often some part of the extension) is directly validated (and, therefore, the intensional model is indirectly validated).

This allows us to distinguish four broad categories of validation strategies:

1. Validation by "Pretty Pictures". This is a typical validation strategy used in E-R modelling and conventional systems analysis. The expert is shown conceptual representations of an intensional model, and asked whether or not this representation accurately reflects the UoD. It is typically difficult for the expert to compute the contents of the sets S_M and T_M from this intensional model, and thus it is difficult to determine whether they map correctly onto the sets S_U and T_U respectively. Essentially, such a representation is difficult to validate because the expert cannot readily perceive its implications.
2. Validation by Consequence Generation. This approach is used in ERAE [6] to validate their predicate calculus specifications:

   "Instead of validating directly the statements produced during modelling, apply first some derivation rule to transform them." [6]

Of course, there are a very large number of logical consequences that can be derived from any given intensional model and it is, in general, impossible to exhaustively generate all possible consequences. Therefore, extreme care must be taken in concluding that the expert's agreement with a certain set of consequences implies the validity of the intensional model from which those consequences were drawn.

The validation strategy used by NIAM [21] [20] is an important restriction of general consequence generation: a significant set of examples (a subset of the extension) is generated from the intensional model, and the expert is asked whether these examples form (part of) a valid extension.

3. Validation by Prototyping. In this approach, an executable prototype is used to provide an extension of (an implementation of) an intensional model. This strategy has two problems. First, the expert must validate at the unfamiliar implementation level, rather than at the conceptual level (where familiar, well-grounded linguistic constructs are available). Second, the expert cannot distinguish between errors in the specification of the intensional model (caused by misinterpretation of the initial data or an ad hoc choice of formal abstractions) and errors in the implementation of that model.
4. Validation by Code Inspection. In the fourth approach, the expert is asked to validate internal (machine) representations of an intensional model. This strategy was used by many early knowledge acquisition tools such as Teiresias. It is typically *very* difficult for the expert to perceive the implications of this model because it is at once abstract (because it is intensional) and unfamiliar (because it is a machine-oriented). Again, it is difficult to distinguish between errors in the initial interpretation and errors in the implementation of that interpretation.

6. Distinguishing UoD Models from Task Models.

Recall that the UoD model contains formalised knowledge about the UoD *per se*, while the task model contains formalised knowledge about some problem-solving process operating on the objects in that UoD.

This suggests that there is a natural, and logically necessary, ordering over the development of these two models. The UoD model must be (at least partially) defined before the the task model can be defined. This is because the task model *must* be defined in terms of the UoD model, but the UoD model can be defined independently of the task model.

We can now introduce an additional level of refinement by partitioning each integrated model in figure 4 into a UoD model component and a task model component. This implies a further partitioning of the processes that produce those models. This is shown in figure 7.

It is interesting to observe the conclusions of Enand, Kahn and Mills in [7] with respect to this distinction: "this distinction [between a UoD model and a task model] facilitates the development of a testing program by allowing validators to develop test procedures appropriate to each set of issues."

7. Using this Process Model for Integration and Automation.

The atomic "design steps" provided by different design methods can be classified by using the previous four distinctions to define the axes of a four-dimensional space. The (initial, intermediate, final) representations used/produced by a given design process can be placed as points in that four dimensional space depending upon which of those distinctions are satisfied by each. The design steps themselves can, therefore, be seen as a arcs from point to point, which are connected to form a trajectory through that space.

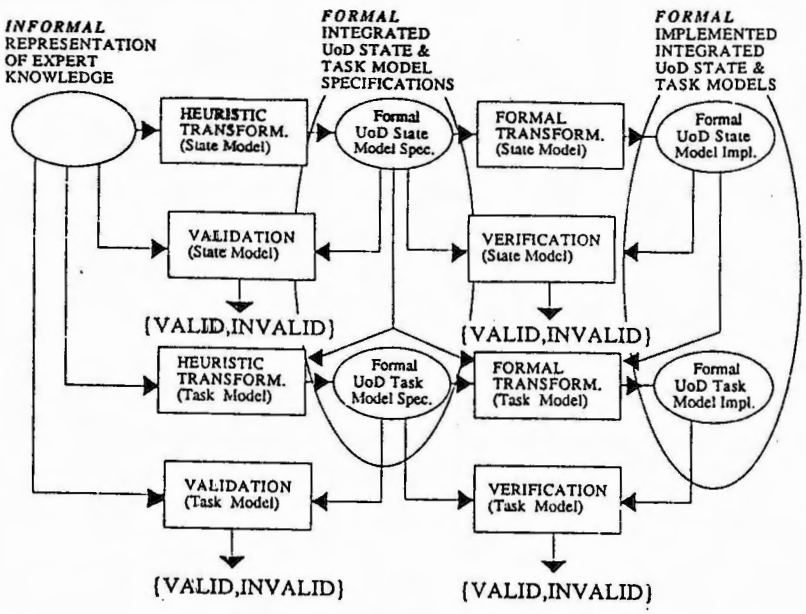


Figure 7: Distinguishing UoD Models from Task Models

We could continue adding additional distinctions to refine the process model further. Each distinction added becomes another dimension in this space, and provides a more precision in comparing/contrasting methods along these various dimensions.

Essentially, the initial abstract model defines the class of "all" knowledge analysis methods. Each distinction subdivides this class into a number of subclasses, each with certain common characteristics. We could continue refining these classes further, by adding distinctions, until the subclass contracts to a singleton set containing exactly one method. In this case, we have arrived at a specification of that knowledge analysis method.

Guiding Integration Efforts. Process A and process B can be said to be *output-equivalent* if their output types are identical, and if the input type of process A is equal to (or a superset of) the input type of process B. In this case, an informal process B could be replaced by a formal process B which specifies the same transformation (but with more prescriptive guidance).

For example, KADS and NIAM could be integrated by taking the UoD model specification (and validation) and implementation (and verification) processes from NIAM, and the corresponding UoD task model processes from KADS. This integrated method has greater scope than NIAM (because it deals with task models) and stronger support than KADS (because the implementation of the UoD model can be formally derived, and, therefore, need not be validated). Of course, it is inherently impossible to replace the heuristic process which arrives at the UoD model and task model specifications. However, NIAM provides some validation processes which KADS currently seems to lack.

Guiding Automation Efforts. Finally, it this model can guide the construction of *support tools* for (partially) automating the knowledge analysis process by the following simple rules:

1. Any formal transformation can, in principle, be automated.
2. Any informal transformation can *not*, even in principle, be automated. At best, it can be supported by heuristics (i.e. "expert systems for expert system design"). At worst, it can be supported by smart editors (i.e. editors which can enforce the specification of the output types). Such a transformation must always be validated.

8. Preliminary Conclusions.

The starting point of this paper is simple:

> Knowledge analysis can be regarded as a process which transforms input knowledge to output knowledge

From this observation, I have constructed a model of knowledge analysis at a very abstract level. This model is, of course, an abstraction which suppresses certain details in order to emphasize others. In particular, this model is a highly technological view of the knowledge analysis process. Other authors (notably Gaines [11], [10]) have provided alternative models of knowledge analysis as a social process (involving social operations such as mimicry and negotiation). It is hardly possible to deny the usefulness of a social perspective in any process involving humans. A social model may be of major use in the *elicitation* phase which prepares the initial information for the knowledge analysis phase. Such an elicitation phase has been neglected in this discussion, but it must occur (the initial information must come from somewhere!).

The most important distinction in the initial model is that between formal knowledge and informal knowledge. This distinction induces a corresponding distinction between formal (algorithmic) knowledge transformations and informal (heuristic) knowledge transformations.

This initial abstract model can be refined by introducing the distinction between specification and implementation. The common assumption here is that this groups all the informal subprocesses together as the "specification" (or "modelling") phase, and all the formal subprocesses together as the "implementation" phase.

At the next level of refinement, I introduced the distinction between transformation steps, which produce model components, and validation steps, which check transformation steps (with respect to some given reference). Every transformation from an informal representation to a formal one should be paired with a validation step. A transformation from a formal representation to another formal representation needs no validation (in theory) because the transformation process can be *proven* to maintain semantic equivalence between input and output representations.

Finally, I introduced the distinction between knowledge of the UoD (which is operated upon) and knowledge of problem-solving processes (which specifies the operations on that UoD). This allows us to partition the "knowledge model" into two smaller models, and also to split some of the various processes. This model defines various classes of methods. These classes were briefly outlined, and some example members of each class given.

Finally, I demonstrated how the simple distinction between formal process and informal process permits us to precisely control the introduction of CASE technology by indicating which method steps can be automated and which can be heuristically supported (or just supported by "editing" facilities). This distinction also suggests ways in which it may be profitable to integrate methods: by replacing an (unnecessarily) informal step in method A with an equivalent formal step in method B to increase the strength of method A, or by replacing "empty steps" in method A with equivalent (formal or informal) steps in method B to increase the scope of method A.

Further Research. This is clearly only a partial answer to the problem of knowledge analysis *methodology* (i.e. studying different knowledge analysis methods and understanding the implications of using a given method). Many questions remain to be answered:

> What is "effective" heuristic support?
>
> What other distinctions are useful in understanding methods?
>
> How can the control dependencies between knowledge analysis subprocesses be described? (I believe that prototyping strategies can be neatly characterized by describing these control dependencies).

An ultimate goal is, of course, the derivation of a metalanguage (and metamethod!) for reasoning about analysis and design decisions involved in a given application of the knowledge analysis process. Such metatechniques may permit us to raise the knowledge analysis process to a more principled and rational level.

References and Bibliography.

1. T. R. Addis, *Designing Knowledge-Based Systems*, Kogan-Page, 1985.
2. T. R. Addis and S. P. Bull, A Concept Language for Knowledge Elicitation, *Proceedings 2nd European Knowledge Acquisition Workshop*, 1988.
3. J. H. Alexander, M. J. Freiling, S. J. Shulman, S. Rehfuss and S. L. Messick, Ontological Analysis: An Ongoing Experiment, *Proceedings AAAI Workshop on Knowledge Acquisition for Knowledge-Based Systems, Banff, Canada*, 1986.
4. I. Benbasat and J. S. Dhaliwal, *Proceedings 3rd Knowledge Acquisition for Knowledge Based Systems Workshop, Banff, Canada,*.
5. J. Breuker and B. Wielinga, Use of Models in the Interpretation of Verbal Data, in *Knowledge Elicitation for Expert Systems: A Practical Handbook*, A. Kidd, (ed.), Plenum Press, New York, 1987.
6. E. Dubois, J. Hagelstein and A. Rifaut, Formal Requirements Engineering with ERAE, *Philips Journal of Research 43*, 3/4 (1988).
7. R. Enand, G. S. Kahn and R. A. Mills, A Methodology for Validating Large Knowledge Bases, *Proceedings 3rd Knowledge Acquisition for Knowledge Based Systems Workshop, Banff*, 1988.
8. E. Falkenberg, H. van Kempen and N. Mimpen, Knowledge-Based Information Analysis Support , *Proceedings IFIP WG2.6/WG8.1 Conference on the Role of Artificial Intelligence in Databases and Expert Systems, Guangzhou, China*, July, 1988.
9. C. J. Fillmore, The Case for Case, in *Universals in Linguistic Theory*, E. Bach and R. T. Harms, (eds.), Holt, Rhinehart and Winston, Inc., 1968.
10. B. R. Gaines, Knowledge Acquisition and Technology, *Proceedings 3rd Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff*, 1988.
11. B. R. Gaines, Second Generation Knowledge Acquisition Systems, *Proceedings 2nd European Knowledge Acquisition Workshop, Bonn*, 1988.
12. N. Johnson, Mediating Representations in Knowledge Elicitation, *Proceedings 1st European Knowledge Acquisition Workshop*, 1987.
13. N. E. Johnson and C. M. Tomlinson, Knowledge Elicitation for Second-Generation Expert Systems, *Proceedings 2nd European Knowledge Acquisition Workshop*, 1988.
14. M. Linster, Kriton: A Knowledge Elicitation Tool for Expert Systems, *Proceedings 2nd European Knowledge Acquisition Workshop, Bonn*, 1988.
15. G. M. Nijssen, On Experience with Large-Scale Teaching and Use of Fact-based Conceptual Schemas in Industry and University, in *Proceedings of IFIP Conference on Data Semantics (DS-1)*, R. Meersman and T. B. Steel Jr, (eds.), Elsevier North-Holland, 1986.
16. A. T. Rappaport and B. R. Gaines, Integration of Acquisition and Performance, *Proceedings 3rd Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff*, 1988.
17. G. Schreiber, J. Breuker, B. Bredeweg and B. Wielinga, Modelling in KBS Development, *Proceedings 2nd European Knowledge Acquisition Workshop*, 1988.
18. M. Sintzoff, Understanding and Expressing Software Construction, in *Program Transformation and Programming Environments*, P. Pepper, (ed.), Springer-Verlag, 1984.
19. J. F. Sowa, Knowledge Representation in Databases, Expert Systems, and Natural Language, *Proceedings IFIP WG2.6/WG8.1 Conference on the Role of Artificial Intelligence in Databases and Expert Systems, Guangzhou, China*, July, 1988.
20. S. Twine, From Information Analysis Towards Knowledge Analysis, *Proceedings 2nd European Workshop on Knowledge Acquisition for Knowledge-Based Systems, Bonn, Federal Republic of Germany*, June, 1988.
21. S. Twine, Towards a Knowledge Engineering Procedure, *Proceedings Expert Systems '88, Brighton, UK*, December, 1988.
22. S. Twine, Representing Facts in KEE's Frame Language, *Proceedings IFIP WG2.6/WG8.1 Conference on the Role of Artificial Intelligence in Databases and Expert Systems, Guangzhou, China*, July, 1988.

Acquisition Of Task–Knowledge For An Adaptive User Interface

Gaby Zinßmeister, Heinz Ulrich Hoppe
Integrated Publication and Information Systems Institute
GMD–IPSI
Dolivostr. 15
D–6100 Darmstadt

ABSTRACT

In order to be able to support the user in his or her problem solving behavior, adaptive systems have to be provided with knowledge of possible task contexts and inference mechanisms to determine adequate help. Existing systems work on a predefined set of task representations. However, there is any number of possible tasks which can never be captured exhaustively by a predefined knowledge base. This paper presents two methods for automatic acquisition of task knowledge. The first method is inductive in that it searches for "similar" task sequences which reoccur in the transcript with a certain frequency. The second method is based on so–called "concepts" which describe the semantics of generic task types in a non–operational way. This approach allows single instances to be recognized and classified according to the semantic concepts they belong to. Those knowledge acquisition mechanisms are embedded in an application environment which provides active and passive help for a UNIX–like file management system.

1 Introduction

Adaptive systems are a topic of growing interest within the field of Human–Computer Interaction. Adaptivity, in this sense, is achieved by providing human–computer interfaces with knowledge, thus enabling the system to respond individually and flexibly to the requirements of a particular user and her/his current task.

One type of adaptivity is the assessment of the task the user is currently performing in order to provide adequate support in achieving her/his goals. According to Card, Moran & Newell (1983), a task may be defined by a top–level goal associated with operators (action primitives) and methods (subtasks with a fixed control structure) to achieve this goal. In order to be able to provide task–oriented user support, it is not only necessary to represent task hierarchies and derive terminal actions from top–level goals, but also to infer the task given a sequence of input actions. The latter is important in order to assess what kind of task the user has performed or is currently performing.

This leads to the questions of an adequate representation and of processing mechanisms for task knowledge. An attributed context–free grammar, in which the user's actions are the terminals and the composite tasks are the non–terminals, is a suitable representation. Based on this representation of task decomposition, the same knowledge base (of grammar rules) can be used for parsing the user's elementary actions and interpreting them in terms of higher level tasks, as well as for task–action mapping. Hoppe (1988) has shown how grammar rules which are very similar to the TAG descriptions of Payne & Green (1986) can be processed to this end by means of a "task–oriented parser". Such a processor was implemented as a bidirectional Prolog program which also allows the generation of actions from given task specifications (therefore called TOP/G: "task–oriented parser / generator").

Given such systems an important consideration is how task knowledge can be acquired automatically or semi–automatically in order, on the one hand, to adapt to the behavior of a specific user, and, on the other hand, to improve the knowledge base in the course of interaction with several users. This paper presents two methods, an inductive and a concept–based mechanism, for learning task knowledge representations from the transcript of a user's session with an application system.

Sections 2.1 and 2.2 treat the representation and recognition of task–knowledge in the form of grammar rules as well as the application environment which has been used. In section 3 the principles of acquiring task knowledge from the dialogue history are discussed. Sections 4 and 5 describe the inductive and concept–based learning methods we propose, and in the conclusion, section 6, the interrelations between the two methods as well as perspectives for the future are discussed.

2 Representation and Recognition of Tasks in an Application Environment

2.1 The Application

Our application environment for task–oriented user support is an active (i.e. system–initiated) and passive (i.e. help on demand) help system for UNIX file management tasks. The FINIX system simulates a subset of UNIX, above all file manipulating operations (such as cp, cat, mv, ..), and provides several user support mechanisms. The support mechanisms rely on task–oriented parsing as performed by the TOP/G interpreter.

FINIX is totally implemented in PROLOG, from the functional kernel of TOP/G and the learning mechanisms up to the PROLOG driven SUNVIEW surface. The interface with some examples is shown in Fig. 2.1–1.

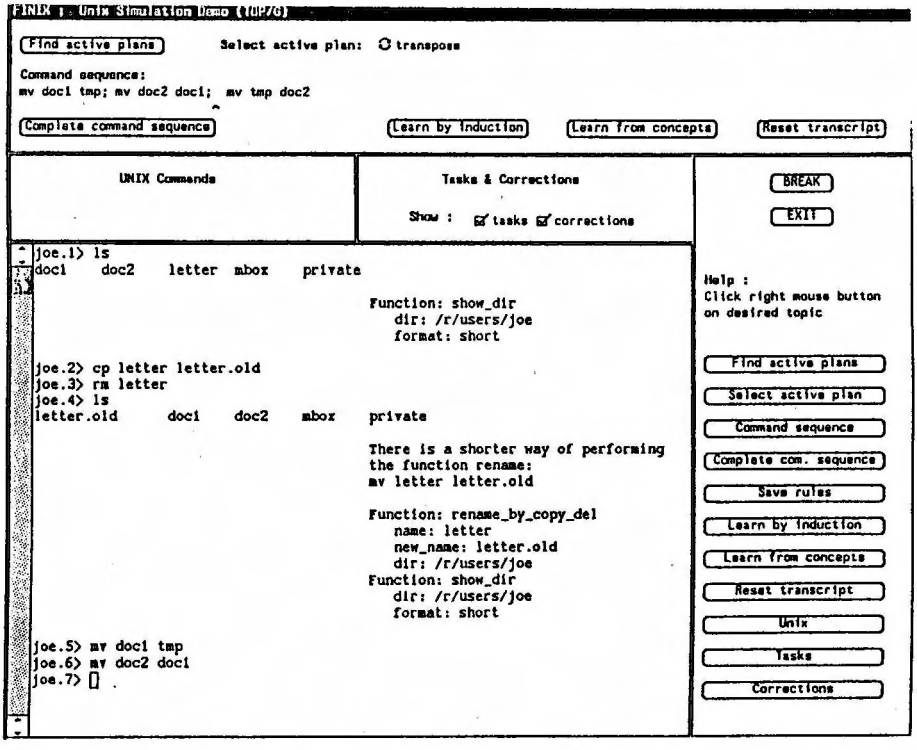


Fig. 2.1–1: FINIX interface

The user can operate in FINIX as in a normal UNIX environment (see UNIX command window). Apart from that it provides a number of support facilities. First the session is analyzed and interpreted as a sequence of more or less complex tasks (see TOP/G window). The "ls" command represents the trivial case of an elementary task corresponding to one command. The more interesting case is the second sequence. The commands "cp" and "rm" form the task "rename_by_copy_delete" with the attribute value pairs listed below. The sequences to be analyzed are delimited by output oriented commands (e.g. the "ls" command). This convention was motivated by the fact that in the UNIX system most of the commands which effect a change in the environment are executed silently. We can assume that as long as the user does not require external feedback, he or she is performing a routine task. Routine task performance is exactly what should be analyzed, e.g. in order to give active help. The user should not be disturbed when using the system in an explorative way.

Based on this analysis, the users' operationalizations of a task can be tested against certain criteria of optimality (e.g. number of operations involved) and, if necessary, a better way of performing the task can be suggested (see TOP/G window). In the above example the task

"rename" can be performed by a single "move" command. During execution of a task the user can retrieve all the active composite tasks which have been initiated and for which all parameters have been specified fully by his/her commands. After clicking the button "find active plans" all currently initiated tasks, which are already completely instantiated, are displayed via menu. In the above example only one plan is active. If an active plan is then selected, the corresponding command sequence is displayed (with a "^" at the current position). The user can choose to complete the task automatically (by clicking the "complete command sequence" button), or to type in more commands by hand. There are, of course, other possible applications such as macro generation of often used "optimal" operationalizations or the application of complex UNDO/REDO strategies and so on, but this should suffice to illustrate the practical purpose of task representation and task analysis.

2.2 Representation of Tasks and their Recognition

The FINIX system is based on the definition of tasks by their operational semantics, i.e. through the operations and parameter constraints on them. Strongly influenced by the task–action grammars of Payne & Greene (1986), Hoppe (1988) developed an attribute grammar representation for task knowledge named LEXITAS, in which the tasks are the non–terminals and the primitives (or elementary tasks) in this case corresponding to single commands are the terminals of the grammar. The grammar rules represent the decomposition hierarchy. Semantic relations can be modeled by attributes and rules for their calculation. At the moment, identity is the only relation between attributes, but it is planned to allow functional (relational) dependencies. Additionally it will be possible to admit lists as attribute values, so that recursive rules can be formulated. Fig. 2.2–1 shows an example for the task "transpose", which means that the contents of two files are exchanged.

Composite Task:

```
transpose ::=  move[ source_dir -> dir1,
                     source_file-> file1,
                     dest_dir    -> new_dir1,
                     dest_file   -> new1]
             +move[ source_dir -> dir2,
                     source_file-> file2,
                     dest_dir    -> dir1,
                     dest_file   -> file1]
             +move[ source_dir -> new_dir1,
                     source_file-> new1,
                     dest_dir    -> dir2,
                     dest_file   -> file2]
```

Elementary Task:

```
move [source_dir, source_file, dest_dir, dest_file]
```

Fig. 2.2–1: LEXITAS representation of a task

The task "transpose" can be performed by three "move" operations. A list of attribute pairs is assigned to each elementary task. The first component of such a pair represents the attribute of the subordinate and the second that of the composite task. The arrow indicates the renaming of an attribute in the context of the higher level task. During the parsing process, the attributes are evaluated bottom–up (synthesized) as well as top down (inherited) in the case of partially

instantiated attribute lists by means of unification. Identical attribute names on the right hand side of the arrows imply constraints. For example, the fact that the source file of the first operation must be the same as the destination file of the second is indicated by the unique attribute name "file1" of the composite task "transpose". In general the components of the decomposition can be composite tasks themselves. Attribute values of elementary tasks can be instantiated in one of the two following ways. Either parameter values of the corresponding command can be associated by so–called "triggering patterns" to the attributes, or values derived from the environment are assigned to the task attributes. This is the basis for mapping user actions onto higher level task units.

The representation of tasks as grammar rules implies that the task–recognition process can be viewed as a parsing problem (Genesereth (1982)), i.e. the decision as to whether a word is in the language produced by the grammar is equivalent to the recognition of a task. The interpretation is performed by the task–oriented parser/generator (TOP/G) which is also used to generate the optimal operationalizations of a task, in order to offer it to the user in case of suboptimal behavior. For further details on the TOP/G see Hoppe (1988).

3 Principles of acquiring task knowledge

A system which provides task–oriented user support can be seen as a sort of "coach". The coach observes what the user is doing and intervenes only if there is an obvious problem or if this is explicitly required by the user. But even the best coach cannot only rely on his previous experience in order to analyze and assess what the trainee is doing. Especially for multifunctional and flexible tools such as text editors, retrieval languages, or operating systems, there exists an arbitrary number of possible tasks which can never be captured exhaustively by a predefined knowledge base. On the other hand, for a particular user there may not be too many complex tasks which frequently occur. Thus it is not only because of the complexity of the potential task knowledge, but also in order to respond to the user's specific needs that we would like to extract the task knowledge dynamically from the dialogue history.

This means that a coach must also be an apprentice in the sense of Mitchell et al. (1985). As far as the requirements for learning algorithms are concerned, we have to face the problem that examples are not directly given but have to be detected in the material, i.e. in the interaction transcript. This leads to learning from observation instead of learning from examples (see Carbonell et al. (1983)). The automated acquisition of task knowledge from a dialogue history generally involves the following three steps:

(1) Detection of subsequences in the transcript which are candidates for being generalized.
(2) Construction of generalized task schemata from the examples.
(3) Evaluation of the new schemata and selection of those to be incorporated into the knowledge base.

This paper presents two automatic mechanisms for the acquisition of task knowledge which differ mainly in step (1): The first method is inductive in that it searches for "similar" task sequences which reoccur in the transcript with a certain frequency. The minimum frequency may be predefined. The second method is based on so–called "concepts" which describe the semantics of general task types in a non–operational way. This approach allows single instances to be recognized and classified according to the semantic concepts they belong to. Concepts are generic in the sense that different operational schemata may be derived from one and the same concept. Both approaches support the finding of hierarchical task structures although this is achieved differently by the inductive and the concept–based method. The introduction of

hierarchical structures interferes with step (2) as far as the global treatment of a set of examples is concerned.

The generalization of a single task sequence is done in the same way for both methods: A new task schema is constructed in the form of a LEXITAS rule which comprises the elements of the sequence as subtasks. Attribute value pairs are replaced by attribute names and constraints are introduced wherever identical values have been found. In order to avoid constraints between subtasks with occasionally identical attribute names, also new symbols have to be introduced. Fig. 3–1 gives an example of the construction of a LEXITAS rule from a task sequence as generated by the task–oriented parser. By default, the name of the new task is the concatenation of the subtask names. In the case of the concept–based approach a semantically adequate naming is possible .

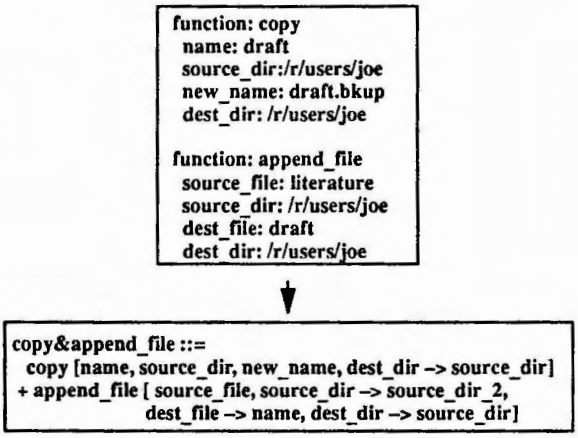


Fig. 3–1: Construction of a LEXITAS rule from a task sequence

The third step, i.e. the evaluation of the results, is currently realized simply by checking the new rules against the existing knowledge base in order to add only rules which are structurally different from existing ones. Therefore, identity of rules is determined irrespective of the naming of attributes and rules.

4 Inductive Knowledge Acquisition

The transcript analysis performed by the inductive mechanism starts with the structuring of the interaction transcript on the level of task types (or functions). An instance of a task as produced by the task–oriented parser is a list of attribute/value pairs. Every instance contains at least the "function" attribute, e.g. with values such as "copy" or "transpose". For the purpose of determining frequently reappearing subsequences, the transcript is reduced to a list of task names or functions. Only maximal sublists which can not be enlarged without violating the frequency criterion are considered. Subsequences of these units are only selected if they also appear independently of the larger context. This is a prerequisite for the parsimonious use of hierarchical structures (see below). E.g., for a minimum frequency of 2 and the initial sequence AGBFCADEFCEGBFCD the algorithm would produce the structured sequence A(GBFC)ADE(FC)E(GBFC)D. Once the transcript has been structured on the functional level, the maximum length of potential compositions (L) is determined.

In the next phase, the complete instances for every task sequence are retrieved from the original transcript and transformed into a task schema as described above. The equivalence classes of rules with the same sequence of functions are partially ordered by the relation "is–more–general–than". A task schema A

$$A ::= A_1 + ... + A_n$$

is said to be more general than another schema

$$A' ::= A_1' + ... + A_n'$$

if function (A_i) = function (A_i')
and constraints (A) subset–of constraints (A').

For each equivalence class of rules with the same function sequence, local maximums with respect to "is–more–general–than" are determined and only these maxima are further considered. I.e. more general examples are preferred to more specific ones. So far, the newly constructed rules are all comprised of pre–existing tasks as sub–units. We would now like to introduce hierarchical structures within the new set of rules (S) according to the following principle:

If $A ::= A_1 + ... + A_n$ in S,
and $B ::= B_1 + ... + B_k$ in S $(k < n)$,

and there is an index i $(0 < i < n)$
so that function(B_j) = function(A_{i+j}) for all j from 1 to k
(i.e. $B_1 + ... + B_k$ forms a subsequence of $A_1 + ... + A_n$)

and constraints (B) subset–of constraints $(A_{i+1} + ... + A_{i+k})$
(i.e. B is more general than the corresponding subsequence of A)

then replace A by $A' ::= A_1 + ... + A_i + B' + A_{i+k+1} + ... + A_n$
where B' is a specialization of B with the necessary additional feature constraints.

In order to introduce subtasks of maximal length, this replacement is performed considering the new rules in decreasing order (k from L to 2). The rules which have already been considered potentially play the role of A whereas candidates for B are taken from the current order k.

Fig. 4–1 gives an example of how this mechanism works. There are two examples of a three–step task which could be interpreted as a "backup_append_delete" (lines 10–12 and 20–22): First a backup copy of some file X is produced, then another Y is appended to X, and afterwards the original Y is removed. Furthermore, lines 15 and 16 contain a similar procedure where the last step is omitted. In the first example the backup directory is identical to the working directory, in the other cases it is not. Therefore, a rule constructed from the first example would bear an additional constraint on the source and destination directories of the copy. But as this rule is less general than the one generated from the last example, it will be dropped. Because it is general enough, the two step version is finally incorporated as a subtask into the more complex structure. Thus, two new rules which are hierarchically nested are constructed from the example sequence.

This method allows us to infer new task knowledge in the form of LEXITAS rules from task protocols produced by TOP/G. The process of identifying an example is guided by certain heuristics: the frequency criterion and the principle of maximality of the sequences to be selected and generalized. Together with the mechanism for introducing subtasks which appear also independent of the higher level context, this implies parsimony with respect to minimizing the

FINIX Input:

```
...
10> cp mbox.bkup
11> cat letter >> mbox
12> rm letter
...
15> cp prog bkups/prog.bk
16> cat testprog >> prog
...
20> cp draft bkups/draft.bk
21> cat literature >> draft
22> rm literature
...
```

Learned Rules:

```
copy&append_file&delete_file ::=
      copy&append_file [new_name, name, source_dir, dest_dir, source_file]
    + delete_file [name -> source_file, dir -> source_dir]
copy&append_file ::=
      copy [name, new_name, source_dir, dest_dir]
    + append_file [source_file, dest_file -> name, source_dir, dest_dir -> source_dir]
```

Fig. 4–1: Example sequence and rules produced

number of new rules as well as the hierarchical complexity of the newly introduced task hierarchy. There are several options to enhance these heuristics. E.g. a combined measure for frequency and length of an example sequence which requires less repetitions for longer sequences could be employed. Furthermore, examples with many constraints, which are in a way more "coherent" than others, could be favoured. This corresponds to the identity heuristics described by Lewis (1988) in the context of analysis–based generalization of procedures.

Although the inductive method leads to the acquisition of task knowledge in a reasonably structured form, this structuring is based merely on syntactic criteria. There is, for instance, no way to determine the type of task which corresponds to a new rule. This is a prerequisite for assigning meaningful names to the new tasks. With the second method, which will be explained in the next section, we are able to overcome these difficulties.

5 Concept–Based Knowledge Acquisition

5.1 The Method

The second learning mechanism was inspired by explanation–based learning and is called concept–based learning (CBL). Fig. 5.1–1 shows an overview of the necessary data and their interrelations.

The domain knowledge consists of task descriptions and semantic primitives. Using semantic primitives, tasks are represented as concepts with a declarative definition of characteristics in the system environment before and after execution of the task. This definition reflects an alternative view of tasks, namely that a task is characterized by its effects rather than by its operations. To

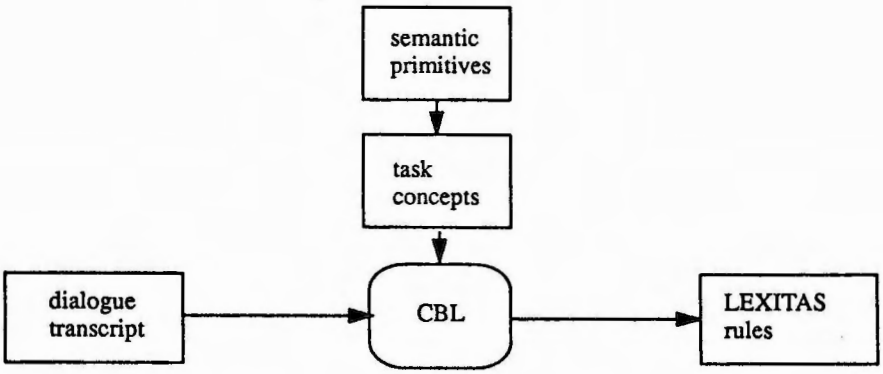


Fig. 5.1–1: Overview

illustrate the representation in the domain knowledge, Fig. 5.1–2 shows a concept description of the task "transpose", of which the LEXITAS rule in Fig. 2.2–1 is an operationalization.

```
concept(transpose,OldEnv,NewEnv,Attr) ::=

        is_file(OldFile1,OldEnv),
        is_file(OldFile2,OldEnv),

        not same_content(OldFile1,OldFile2),

        is_file(NewFile1,NewEnv),
        is file(NewFile2,NewEnv),

        equal_dir_&_name(OldFile1,NewFile1),
        equal_dir_&_name(OldFile2,NewFile2),

        same_content(OldFile1,NewFile2),
        same_content(OldFile2,NewFile1),

        Attr = [attr(file1,OldFile1),
                attr(file2,OldFile2)]
```

Fig. 5.2–1: Example of a declarative concept description

The verbal interpretation of this concept is: "There are two files (OldFile1,OldFile2) in the system environment before the execution of the task (OldEnv). They do not have the same content. There are two identically named files in the environment after execution of the task (NewFile1, NewFile2 in NewEnv) and the contents have been interchanged."

The attributes describe the external attributes of the task; in this example the two files which are transposed. Thus meaningful attribute and rule names can be passed on to the learned rules. The semantics of the primitives should be clear from their names. Important is that there is no mention of operations in this representation. It consists solely in the definition of what is done, and not how to do it.

The input for the learning algorithm is an extended transcript of the user dialogue on the elementary task level. Beginning with the initial system environment, the elementary tasks of the user alternate with the environments resulting from the application of the elementary task to the previous one. And, as is the case with the inductive learning algorithm, generalized operationalizations in the form of LEXITAS rules are induced. For this purpose, the varying states of the environment in the transcript are examined in pairs, in order to determine the validity of a concept. These pairs are chosen in increasing order, i.e. pairs surrounding fewer operations are examined before those which are further apart. Therefore operationalizations capturing few elementary tasks are learned before larger (possibly composite) ones.

If a pair is found for which the relationship defined in the concept holds, the corresponding task is "recognized". Operations which do not contribute to the effect of the task are weeded out of the sequence of elementary tasks. Those remaining are generalized as a LEXITAS rule by means of the same mechanism used in the inductive learning method. Learned rules are added to the rule base if no equivalent one already exists (see chap. 4.).

If there are no irrelevant elementary tasks the transcript is consolidated, i.e. the sequence of elementary tasks and the environment states between them are replaced by the corresponding composite task. Thus hierarchically composed tasks can be learned in one run through the dialogue transcript. Continuing the above example, Fig. 5.1–3 shows the learning process. The transcript need not contain the entire states of the environment. Skeletons of them containing those parts referred to in the command sequence will suffice.

The dialogue transcript consists of a sequence of elementary tasks which are applied to an initial system environment. The CBL method looks for valid concepts and reveals the relevant operation sequences (i.e. the exemplary instance), which are generalized to a LEXITAS rule. This description is simplified, since in reality it is an iterative method, as during the recognition process the transcript is manipulated and the rules are generated incrementally.

5.2 Concept–Based Learning versus Explanation–Based Learning (EBL)

Although EBL and concept–based learning (CBL) have very much in common there are fundamental differences.

The main characteristics of EBL are the following: It needs a strong domain theory and one exemplary instance of an identified concept. First it proves that the example given by the user is an instance of the specified concept which the algorithm already knows. Then this proof tree is generalized and a new rule is formulated for the generalized proof. The rule is an abbreviation of the proof tree assigning the frontier to the root (see DeJong & Mooney (1986), Witten & MacDonald (1988), Michalski et al. (1983)).

So both, EBL as well as CBL, use non–operational concept descriptions as domain knowledge and one exemplary instance as basis for the generalization. The learned knowledge is operational (even though in a different sense) and more efficient in use than the domain knowledge. But there are characteristic differences.

In CBL the learned and the domain knowledge have principally different representations; the former is operational in the form of LEXITAS rules representing a decomposition hierarchy, the latter is non–operational, defining the effects of the tasks. EBL, on the other hand, uses domain knowledge containing a decomposition hierarchy (which can also be viewed as a context–free attribute grammar), implicitly including the set of operational primitives to be learned. In this respect the LEXITAS rules learned in CBL are very similar to the domain knowledge of EBL.

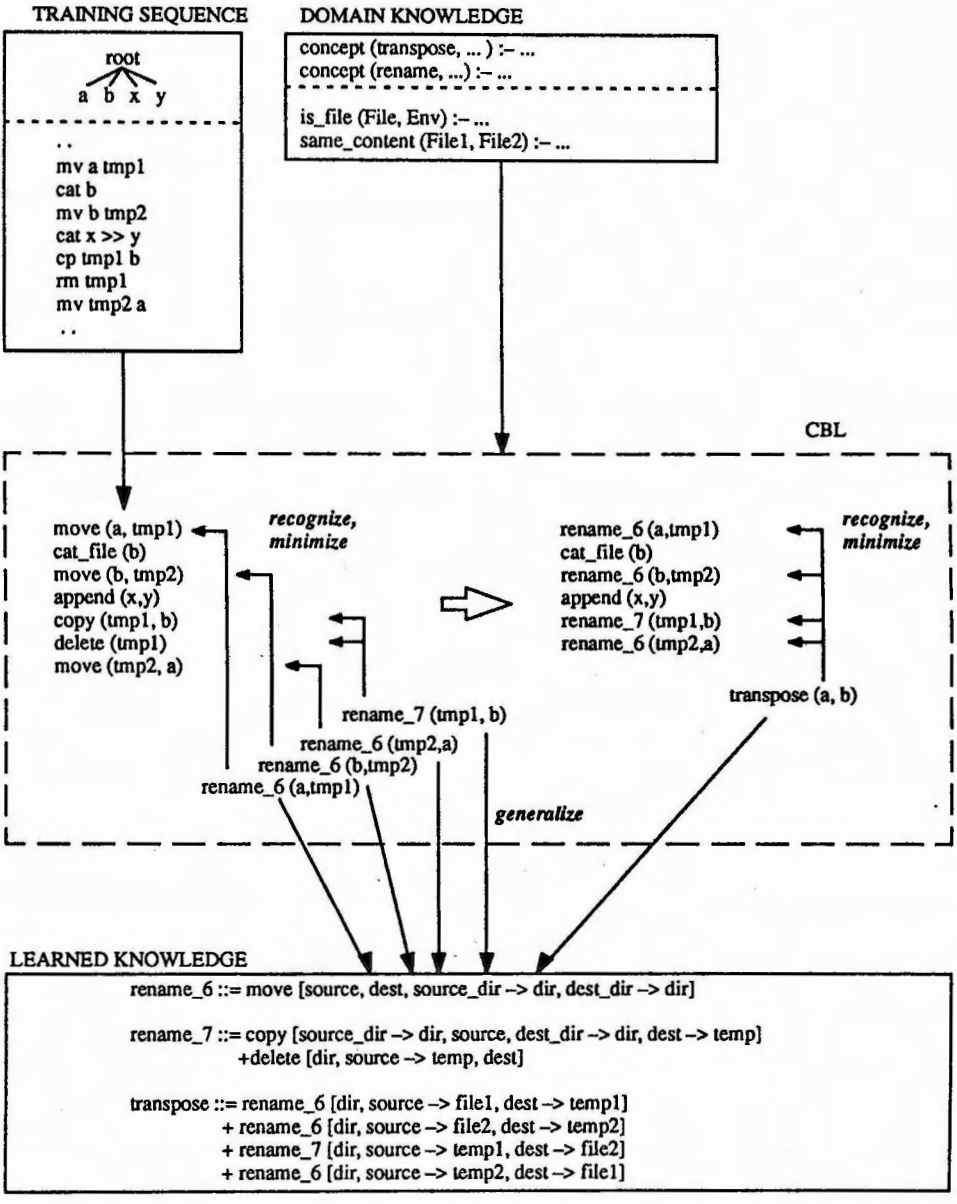


Fig. 5.1–3: An example of learning

Note that there is a difference in the interpretation of the term "operationalization" in the two methods. In EBL the operationalization of a concept is the corresponding set of operational primitives. In contrast, in CBL the LEXITAS–rules, which belong to one concept and define a decomposition hierarchy including the operational primitives, i.e. the elementary tasks, are the operationalization.

In the CBL domain knowledge, concepts are represented on a more general level, independent of operations, and therefore capture all possible operationalizations the user might apply. As a consequence, the concepts serve only to recognize those operations (throughout an entire session) which cause such changes in an environment as are specified in a concept description. In other terms, it reveals the exemplary instance and the corresponding concept, whereas in EBL one exemplary instance and the corresponding concept are given.

In CBL, the generalization is based on the recognized operations of the training instance and without using the domain theory, whereas in EBL the derivation tree constructed from the domain knowledge is generalized. Fig. 5.2–1 illustrates that the task recognition and the generalization of the operations work in "different worlds" with different data structures.

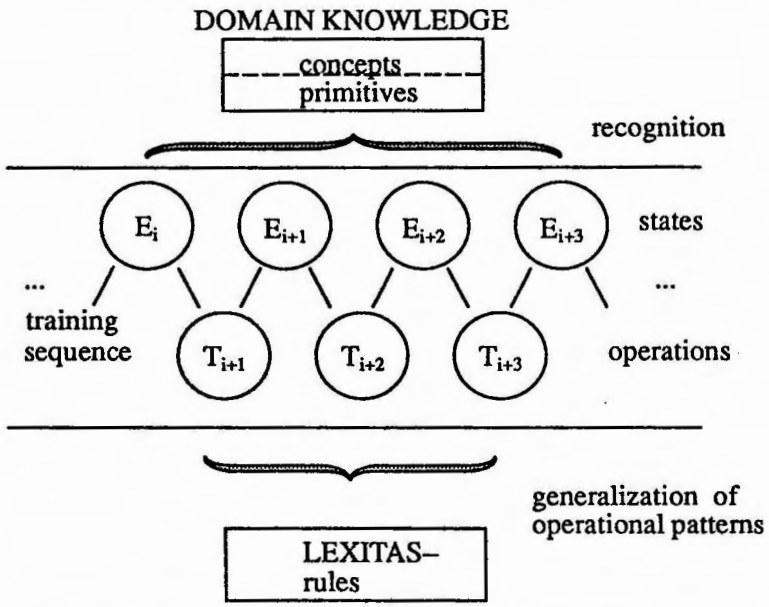


Fig. 5.2–1: separation of task recognition and generalization in concept–based learning

5.3 Types of generalization

To characterize the generalizations performed in the concept–based method we will apply the classification, used by DeJong (1988) to describe EBL, to the features of CBL.

a) **Irrelevant Feature Elimination**

In EBL the operational primitives are so–called "features". Therefore, irrelevant feature elimination means that operational primitives not mentioned in the explanation are eliminated. In

the case of CBL, the elementary tasks in the dialogue are the operational primitives. The elimination of irrelevant primitives is performed by dropping those elementary tasks which do not contribute to the concept (see example in Fig. 5.1–3).

b) Identity Elimination

Unnecessary dependency on particular objects is removed. Identity elimination consists in the parametrization of specific components of the explanation. In CBL, specific arguments of the operationalization are parametrized in the learned rules. For example, given the task "rename", i.e. a file is given a new name or a new location in the directory hierarchy, and given a suboptimal operationalization consisting of the two commands:

```
cp a b
rm a
```

we can parameterize specific components of the exemplary instance, whereby the identity relations between the parameters must be kept. In this example constants are replaced by variables with identical variables describing the same object. The corresponding rule would look like this:

```
rename ::=   copy   [ source_dir  ->  dir,
                      source_file ->  file,
                      dest_dir    ->  dir,
                      dest_file]
           +delete [ dir,
                      file]
```

c) Operationality Pruning

In EBL operationality pruning means elimination of easily reconstructable sub–explanations from the explanation. Internal nodes of the proof tree may be operational too, so their sub–trees can be dropped. In contrast to the usual situation in EBL, CBL makes a clear–cut distinction between operational and non–operational concepts. In the field of task knowledge operational is equivalent to executable. Therefore the learned rule representations for tasks are operational per definition. They are efficient representations of elementary task sequences and those in turn are representations of system commands. By introducing macro commands with corresponding elementary tasks, subtrees of the decomposition hierarchy can be replaced.

d) Structural Generalization

The preceding types of generalization referred to generalizing from training instances to the learned concept. "Structural generalization", in contrast, means that the internal structure of the explanation itself is altered, possibly making the derived operational concept more general. In CBL, we do not build an explanation or a proof tree as a basis for the learning and generalization process. Thus, a direct application of the classification seems not appropriate. Nevertheless some aspects should be taken into account.

In contrast to EBL, in CBL hierarchically structured concept representations are learned. In some ways the decomposition rules are similar to the domain knowledge of EBL. "Number generalization" in EBL means that particular sub–explanations can be arbitrarily replicated. So the representation of loops must be introduced into the formalism. In CBL a similar problem is to generalize from the number of repetitions of a subtask, which implies the augmentation of the formalism with recursive rules. For example, suppose we want the system to learn rules representing multiple changes to the current directory. If the training sequence gives three "change directory" commands, the system will learn the following rules:

```
mult_set_dir_7 ::=
        set_dir [source_dir, dest_dir -> dir1]
      + set_dir [source_dir -> dir1, dest_dir]
mult_set_dir_12 ::=
        set_dir [source_dir, dest_dir -> dir1]
      + mult_set_dir_7 [source_dir -> dir1, dest_dir]
```

The system is now able to recognize two and three–step changes to the current directory. If the rule names are not unique and rules referring to the same concept have identical names, the rules generalize from the concrete number of repeated subtasks. Suppose in the above example that the suffixes "_7" and "_12" are dropped. Then the system will be able to recognize arbitrarily long sequences of "change directory" operations. In order to generate recursive rules it is sufficient to drop the suffixes. For "disjunctive augmentation" (which means that alternative decompositions for one and the same (sub)task can be specified) the same mechanism would work as well.

Due to some specific problems we have as yet not made use of these types of generalization. The main problem is the fact that different operationalizations of one and the same concept may produce different side–effects in addition to the "significant changes" which are required by the concept. The use of one of these operational schemata in a higher level task schema could be bound to its specific side–effects. But in the new higher level schema, there would be no way of distinguishing between the different versions of the subtask if they were not indexed. This problem could be overcome by the association of applicability conditions with the decomposition schemata.

A further type of generalization which we have not attacked yet is "temporal generalization". This designates the abstraction from the specific order of operations given in the training instance, since there might exist a different sequence of the same operations which would work as well. A brute force method would be to generate all permutations, propagate the environment over them and select those where the primarily detected concept still holds. Improvements to this approach could be made by applying heuristics to cluster independent operations and therefore reducing the set of meaningful permutations.

6 Interrelationships between the inductive and the concept–based learning mechanisms

There is semantic evidence for the knowledge learned by the concept–based method. The generalization process itself guarantees that the generalization specifies a meaningful concept, whereas in the inductive learning mechanism the generalization is based only on empirical evidence, whereas there is no guarantee that it will be useful or even semantically well–founded. This touches on the question of the evaluation of the new knowledge (see chap. 3). For the inductively learned rules semantic evidence could be introduced by human judgement. From the point of view that the learned knowledge should serve to support users performing routine tasks it seems reasonable to empirically evaluate the knowledge learned by CBL, so that only frequently reappearing tasks are incorporated into the knowledge base, even though one single occurrence suffices for learning the operationalization.

Thus the concept–based method is able to recognize and classify arbitrary operationalizations of tasks, and is limited by the predefined set of declaratively represented concepts. Because the inductive learning mechanism works only on the dialogue transcript as a basis of frequently reoccuring patterns, it is principally possible that the learned rules are operationalizations of new task concepts. To learn corresponding concept representations, descriptions of the effects of the

learned LEXITAS rules must be deduced. After clustering rules with "similar" descriptions, a further learning mechanism of the type "learning from examples" must be applied to each cluster. In any case a validation of the semantic evidence of the so–derived concepts must follow.

7 Perspectives for Future Work

Future work on the FINIX system and the two methods will take the following directions:

First, the efficiency of the concept–based algorithm will be improved. If the representation of the environment states in the transcript is filtered by the executed tasks, i.e. only those parts of the environment are stored which have been referred to by the user commands, time and storage needed by CBL will be reduced. Since users work "locally", in other words use only files in relatively small connected parts of the environment, such a filter promises to be an effective improvement of the algorithm. This would even make it feasible to use CBL during an interactive session as a sort of higher–level plan recognizer.

Second, present limitations of the concept–based method must be overcome. As mentioned in chapter 6 it is necessary to develop a mechanism to enlarge the set of task concepts, in order to introduce new tasks and to get a more realistically sized knowledge base.

Furthermore, the LEXITAS representation should be extended to allow more general attribute dependencies (cf. 2.2). By introducing more complex relationships such as list predicates, arithmetic relations or constant values, the class of tasks which can be modeled can be enlarged. Especially the definition of recursive tasks can be supported by list–type attributes. This expansion raises the problem that such complex relationships must be deduced from the exemplary instance. This type of learning can be classified as function induction in accordance with Witten & MacDonald (1988). Additionally, a model must be developed to pass constraining relations already definable in the declarative concept descriptions to the learned rules. A further expansion of the LEXITAS formalism should comprise the definition of preconditions which characterize the special applicability constraints on the operationalization. Obviously, if some of the effects of a task already hold before beginning an operation sequence, several operations can be left out. Therefore, if a rule is learned by the concept–based method, such "preholding" effects must be passed from the concept to the rule.

We wish to thank Rolf Plötzner for his work on the development of the generalization algorithm and Bernd Kostka for the complete redesign and improvement of the UNIX simulation and for the implementation of the SunView surface to FINIX.

8 Bibliography

Carbonell, J.G., Michalski, R.S. , Mitchell, T.M., 1983
"An Overview of Machine Learning"
In: Michalski, R.S., Carbonell, J.G., Mitchell, T.M. (Ed.), 1983 Machine Learning: An Artificial Intelligence Approach, M. Kaufmann Publishers, Los Altos, Calif.

DeJong, G. (1988)
"Some Thoughts on the Present and Future of Explanation–Based Learning"
In: Proceedings of the European Conference on Artificial Intelligence ECAI 1988, Munich, Pitman Publishing, pp. 690–697

DeJong, G.; Mooney, R. (1986)
"Explanation–Based Learning: An Alternmative View"
In: Machine Learning, Vol.1, p. 145–176

Genesereth, M.R. (1982)
"The role of plans in intelligent teaching systems"
In: Intelligent Tutoring Systems; ed. D. Sleeman, J.S. Brown, Academic Press
Hoppe, H.U. (1988)
"Task–Oriented Parsing – A Diagnostic Method to be used in Adaptive Systems"
In: Proceedings of the CHI'88, Washington, pp. 241–247
Hoppe, H.U.; Plötzner, R. (1989)
"Inductive Methods for Acquiring Task Knowledge in Adaptive Systems"
paper presented at the 8th Interdisciplinary Workshop on "Informatics and Psychology", Schärding, May, 1989
Lewis, C. (1988)
"Why and how to learn why – Analysis–based generalization of procedures"
In: Cognitive Science, Vol. 12, pp. 211–256
Mitchell, T.M.; Keller, R.M.; Kedar–Cabelli, S.T.; (1986)
"Explanation–Based Generealization: A Unifying View"
In: Machine Learning, Vol.1, pp. 47–80
Mitchell, T.M.; Mahadevan, S.; Steinberg, L. (1985)
"Leap – A Learning Apprentice for VLSI design"
In: Proceedings of the IJCAI '85, Morgan Kaufmann, Los Altos, Ca
Payne, S.J.; Green, T.R.G (1986) May 18, 1989
"Task–Action Grammars: A Model of the Mental Representation of Task Languages"
In: Human–Computer Interaction, 1986, Vol. 2 pp. 93–133
Witten,Ian H.; MacDonald, Bruce A. (1988)
"Using concept learning for knowledge acquisition"
In: International Journal of Man–Machine Studies (1988), Vol 29, No.2, pp. 171–196
directions

EKAW-89

Integration of Machine Learning and Expertise Transfer Techniques

EKAW-89 Tutorial Notes: Machine Learning
Paris, France, July 1989

Jean Gabriel GANASCIA

LAFORIA, Université Pierre et Marie Curie
Tour 45-46, 4 place Jussieu, F-75252, Paris CEDEX 05, FRANCE

INTRODUCTION

The goals of machine learning are to develop computational theories of learning and to build learning machine. From this last point of view, machine learning can be seen as a process transforming sources data (observations, traces of computation, teacher remarks...) into organized and usable knowledge. This process may involves many differents methodologies:

- Learning concepts descriptions, productions rules or decision tree from examples
- Learning control strategies and heuristics through experimentation
- Building hierarchies of concepts
- Connectionist network
- Genetic algorithm
- ...

Those methodologies may themselves involve different operations:

- deduction: $\alpha \Rightarrow \beta, \alpha \mid\!- \beta$
- induction: $\alpha(A) \Rightarrow \beta, \alpha(B) \Rightarrow \beta \mid\!- \alpha(x) \Rightarrow \beta$
- abduction: $\alpha \Rightarrow \beta, \beta \mid\!- \alpha$
- analogy: $\alpha \Rightarrow \beta, \alpha' \approx \alpha \mid\!- \beta' \approx \beta$
- ...

According to [Michalski 86] three dimensions can help to classify machine learning technics: *learning strategies*, *knowledge representation* and *application domains*.
Learning strategies:

- Rote learning
- Learning form instruction
- Learning by deduction
- Learning by analogy
- Learning by example
- Learning by observation and discovery

Knowledge representation:

- Coefficient on mathematical expressions
- Decision tree
- Production rules
- Logical expressions
- Semantic nets, frames or schemas
- Taxonomies
- Programs or procedures
- ...

This tutorial provides an introduction to some existing methodologies that make computers learn. Since we are mainly concerned with Knowledge Acquisition, we shall concentrate here on practical achievements of machine learning applied to Knowledge Acquisition. We shall present operations involved in those learning systems and their limits.
One can distinguish two practical topics for machine learning:

- Building knowledge bases from data
- Performance improvement

BUILDING KNOWLEDGE BASES FROM DATA

It is to generalize a set of observation in order to acquire usable and organized knowledge. There exist two main strategies of generalization:

- Top-down
- Bottom up

Top-down strategies: generalization from the general to the specific.

The building up of decision trees with the Quinlan method [Quinlan 83] and its numerous additions is a good representation of this approach. To summarize, we start from:

– A set of classes exclusive from one another: $C_1 ... C_k$

– A set of examples $E_1 ... E_n$ for each class C_i

Construct a decision tree to classify all the examples E_i

Each example is described by an attribute value vector, for instance:

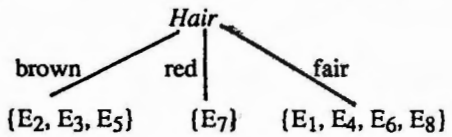
	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
Size	short	short	tall	tall	tall	short	tall	tall
Hair	fair	brown	brown	fair	brown	fair	red	fair
Eyes	blue	blue	brown	brown	blue	brown	blue	blue
Class	+	-	-	-	-	-	+	+

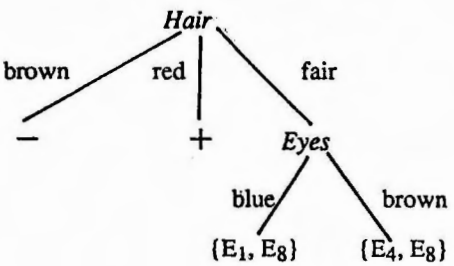
Learning algorithm to construct a decision tree for the set of example S:

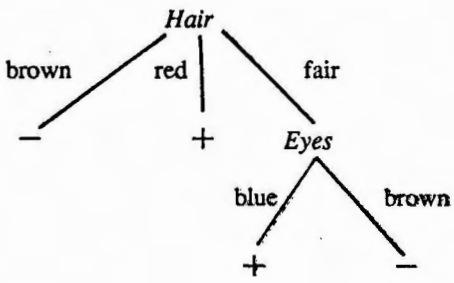
If all examples in S are of the same class C

Then label the corresponding node with C

Else - select 'most informative' attribute A with values $v_1, v_2, ..., v_n$.
- partition S into $S_1, S_2, ..., S_n$ according to value of A.
- construct subtree $T_1, T_2, ..., T_n$ for $S_1, S_2, ..., S_n$.







Refinements:
- 'Most informative' attribute: information theoretic criteria, errorcost, ...
- Handling incomplete data
- Handling ordinate attributes
- Stopping criteria
- Pruning trees (noisy data)
- ...

Advantageous:
- Cope with a lot of examples
- Deal with noisy data

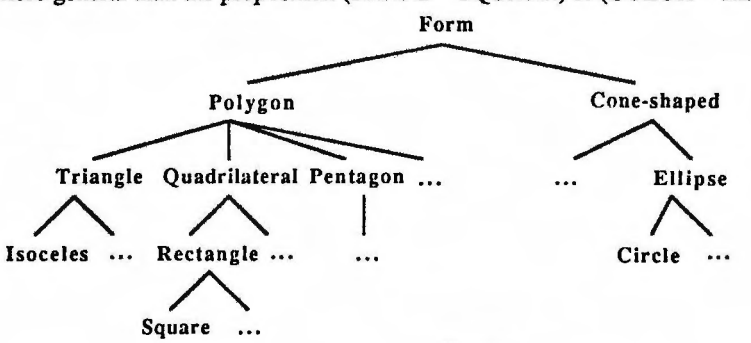
Critics:
- No accounting for the description language semantics.
- Learning heuristics based on numerical functions.

Top-down strategies: generalization from the specific to the general.
Michalski INDUCE system [Michalski 83] is the most typical example of this type of approach and can be schematically expressed as follows:
Given
- A concept **C**
- A set of examples $E_1...E_n$ and of counter-examples $CE_1...CE_p$ of **C**

Stage 1: Construction of a generalization of $E_1...E_n$ discriminating $CE_1...CE_p$. Call it E_g

Stage 2: Construction of the rule $E_g \rightarrow C$

Many papers (Cf. [Michalski 83], [Mitchell 82], [Plotkin 70], [Kodratoff & Ganascia 86], etc.) have shown the difficulties to elaborate a consistent theory of generalization. We shall not dwell on this aspect, except to recall a few aspects of the generality relation, and, more specifically, their formal translation:
- **Constant variabilization:** if x is a variable, A a constant, P a predicate and $P(t_1, t_2, ..., t_n)$ an atom then $P(x, t_2, ..., t_n)$ is more general than $P(A, t_2, ..., t_n)$.
We shall note it as $P(x, t_2, ..., t_n) \leq P(A, t_2, ..., t_n)$.
- **Use of the dropping rule:** if P1 and P2 are two propositions, P1≤P1 & P2.
- **Climbing in a conceptual hierarchy:** given an elementary conceptual hierarchy to explicit relations of generality between plane figures (Cf. figure). In such a hierarchy, POLYGON<SQUARE and thus the proposition (SHAPE = POLYGON) & (COLOR = RED) is more general than the proposition (SHAPE = SQUARE) & (COLOR = RED).

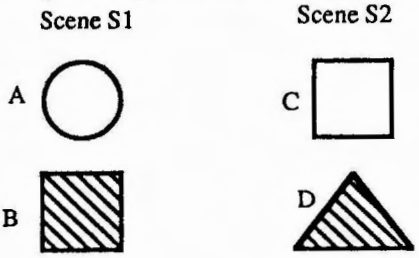


figure

– **Extension of boundaries** in the case of valued attributes on ordered sets: if SIZE is an attribute which has values over the set of integers, (SIZE ≥ 7) is more general than (SIZE ≥ 37) and thus, (SIZE ∈ [-7, 7]) ≤ (SIZE ∈ [-5, 5]).

However, even in simple examples, the number of possible generalizations is very high. Let us mention for the sake of illustration some of the possible generalizations for the scenes S1 and S2 (Cf. figure):

- There are two objects on top of one another
- The lower object is a polygon
- There are two objects and one of them is a square
- There is a striped polygon
- ...



Figure

In front of this large number of potential intuitive generalizations, it is indispensable to chose the most adequate, that is that, which in a given context will express best the common and relevant characteristics of the examples. Also, whatever the formal theory of generalization chosen, an order of priority as to the potential generalizations must be indicated.

Although this technique has been confirmed by major successes, it can be criticized on many respects. In fact, generally, the rules constructed cannot be used directly by an inference engine, and whenever they are, it is not without a major impoverishment of the resulting expert system. Among the causes of such an impoverishment, we may quote:

– Lack of rule chaining: A → B, B → C.
– Difficulties, or even impossibility, to introduce uncertainty.
– Lack of consistency and economy for the knowledge base: the rules are considered as isolated from one another.
– Last, the frequent use of heuristics during the generalization process which is not justified as these heuristics have no semantics.

CHARADE : systems of rules learning

Designed to facilitate the man to machine expertise transfer, CHARADE automatically builds up knowledge bases from:

– A *description language* covering a set of typed attributes and axioms expressing the domain semantics.
– A *set of examples* described in this description language
– The description of the *expert system functionalities* for which the knowledge base is to be constructed.

CHARADE operates by empirical detection of regularities. Its main characteristics are that:

– It takes into account the semantics of the domain expressed as axioms.
– It can simultaneously generate certain rules and uncertain rules modulated by a plausibility coefficient.
– It constructs systems of rules which can be directly used by commercial inference engines, rather than simply isolated rules.
– It can translate the funtionalities of the expert system to be obtained into generalization heuristics. This is the main originality of CHARADE which provides the generalization heuristics with clear semantics.

This is the most original aspect of CHARADE as, thanks to such properties, it establishes clear semantic for the generalization heuristics that are usually applied, at the same time that it introduces new heuristics. Thus, these properties become the operational characteristics of the expert system that we want to obtain. In fact, to build up a knowledge base, one must refer to the finality of the desired expert system. Otherwise, it would be a collection of rules, many of which useless, some corresponding to random coincidences, others having no practical interest

whatsoever. These characteristics are used, sometimes, to create knowledge acquisition tools to verify the relevancy of the rules given by the user. The originality of the CHARADE system, is to translate these characteristics into constraints for the exploration procedure of the description space. As an illustration, we present below, some of these properties:

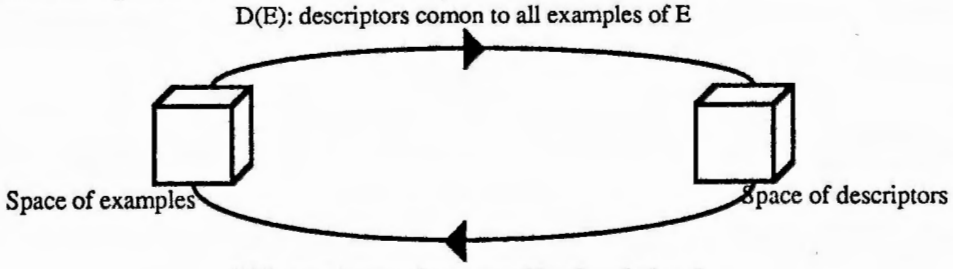
– **Goal** of the rules system (diagnosis, classification...). In the case of classification systems, if $C_1,..., C_n$ are classes, whenever a rule of the type $a \rightarrow C_i$ is generated, rules more specific than a (i.e. a&...) are **useless.**

– **Minimum number of examples** covered by the rule premise: if card(C(a))≤v then rules more specific than a (i.e. a&...) are **useless** as **card(C(a&...))≤v**

– **A-priori structure of the rule system:** when an expert system is constructed, the structure of the system of rules that we want to generate is known a-priori. Thus, for a therapeutical aid, whether in medicine or vegetal pathology, one knows beforehand that the rules go from the symptoms to the disease and then from the disease to the relevant therapy. This is what is reflected in the right diagram.With this hierarchy, it is then possible to leave aside the exploration of the descriptions which correspond to a therapy and the study of regularities going from the diseases towards the symptoms of from symptoms to symptoms.



To construct knowledge bases CHARADE [Ganascia 87] empirically looks for regularities which are present in the learning set. To do so, the set of parts of the learning set and the set of descriptor conjunctions are represented by two Boolean lattices. Besides the representation economy that they offer, the properties of the lattices are used in the learning process to facilitate the detection of regularities. In fact, a regularity corresponds to a correlation empirically observed in the learning set. If all examples that have a descriptor d_1 in their description also have the descriptor d_2, it is possible to induce that d_1 implies d_2 in the learning set. Thus, these regularities must then be detected. Two functions D and C are used. The first one, D, goes from the descriptors lattice to the examples lattice. To each description, it associates the set of examples of the learning set covered by this description. The other, C, relates to each subset of the learning set, i.e., to each element of the examples lattice, the set of descriptors present in the description of all the examples. These two functions are represented below:



Example: Let us imagine a learning set made of three examples E_1, E_2, and E_3:

$E_1 = d_1\&d_2\&d_3\&d_4$

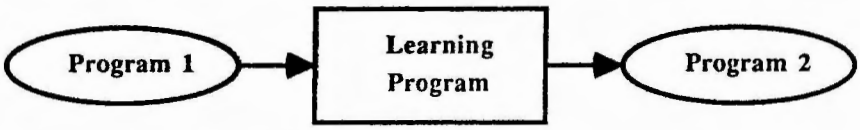
$E_2 = d_1\&d_2\&d_4\&d_5$

$E_3 = d_1\&d_2\&d_3\&d_4\&d_6$

An empirical regularity can be detected between $d_1\&d_2$ and d_4, in fact, all the examples described by $d_1\&d_2$ have also d_4 in their description. This is obtained with D and C as follows:

$DoC(d_1\&d_2) = D(\{E1, E2, E3\}) = d_1\&d_2\&d_4$

PERFORMANCE IMPROVEMENT



Program 2 solves the same problems as program 1, but it is more efficient
We shall study two method:

- Learning solving heuristics
- Explanation based-learning

Learning solving heuristics: the LEX example [Mitchell 1978]
Given:

- Heuristic problem solver without heuristics
- Set of problem-solving operators
- Language for describing heuristics

Determine:

- Heuristic that should be applied to solve any given problem

Operators:

Op7: $I(\cos(x)\,dx) \rightarrow \sin(x) + e$

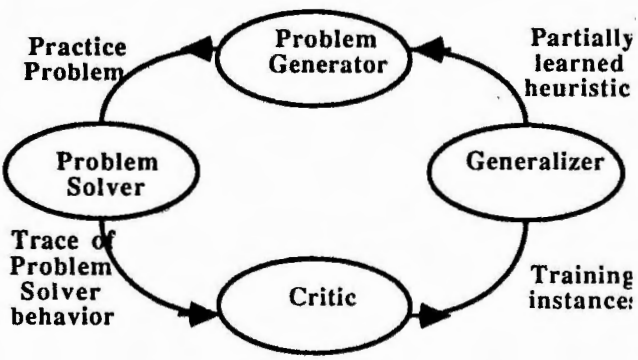
Op9: $f_1(f_2 + f_3) \rightarrow f_1f_2 + f_1f_3$

Op12: $I(u\,dv) \rightarrow uv - I(v\,du)$

A learned heuristic:

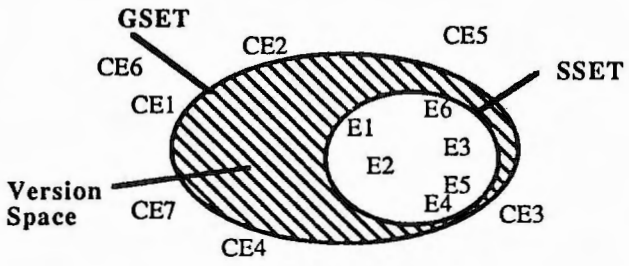
$I(x\ trig(x)\,dx) \rightarrow$ TRY Op12
WITH u=x
WITH dv = trig(x) dx

Design of LEX:

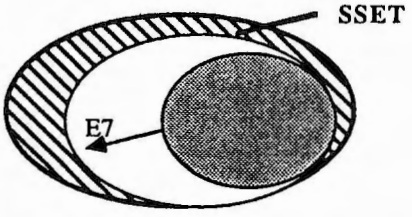


How to chose the appropriate generalization ?
How to learn incrementally ?

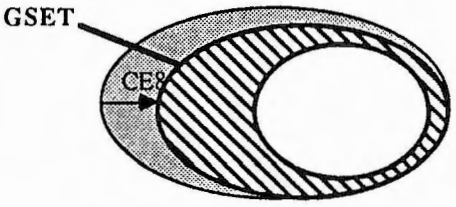
Version space



Adding an example



Adding a counter-example



Explanation-Based Learning (EBL)

Creation of an efficient operator for a class of examples from a single training example.
The main idea is to use knowledge of the domain to understand how the training example is solved, and then to generalize the corresponding proof.
Family of Explanation-Based Methods

- Explanation-Based Learning
- Explanation-Based Generalization
- Analytical Generalization
- Chunking
- Macro-operators

Overview of EBL [Mitchell 86]
Given the following four types of knowledge:

- a *goal concept*: the concept to be learned
  For instance suicide(x)

- a *training example* which is an instance of the concept:
  Depressive(John)
  Buy(John, O1)
  Gun(O1)
  Tall(John)
  Eyes_color(John, Blue)

- *operationality criterion*: language restriction
  Easily computable predicates
  Depressive(x), Buy(x,y), Weapon(z)

- *Domain theory*:
  R_1 : Kill(x,x) $\supset$ Suicide(x)
  R_2 : Hate(x,y) & Own(x,z) & Weapon(z) $\supset$ Kill(x,y)
  R_3 : Depressive(t) $\supset$ Hate(t,t)

R_4 : Buy(u,v) ⊃ Own(u,v)

R_5 : Gun(w) ⊃ Weapon(w)

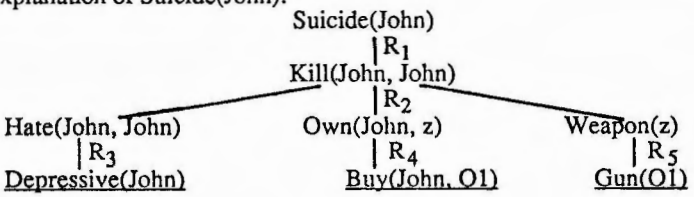
R_6 : Dagger(w) ⊃ Weapon(w)

...

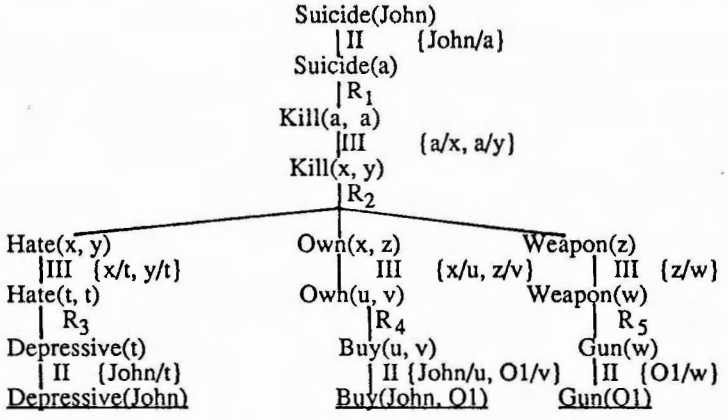
Find a generalized operational description of suicide
Steps:

1- Use the domain theory to prove that the training example is an instance of the goal concept
2- Generalize and contract the proof.

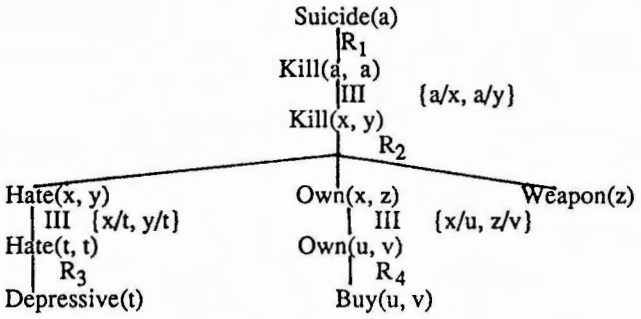
1- Building explanation of Suicide(John):



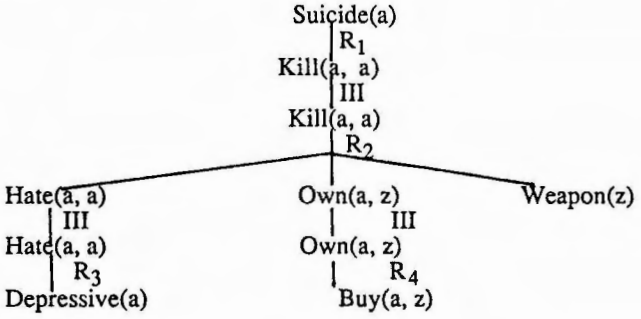
2- Generalizing the proof
Tall(John) and Eyes_color(John, Blue) are not present in this proof, so they will not be present in the solution.
We have now to distinguish general unification (we shall note them II) which are required to match the rules, from specific one (we shall note them III) which correspond to the example description.



3- Pruning the proof tree by rejecting unification specific to the example (i.e. {John/t} or {O1/v}) and subtree of a node verifying the operationality criterion (i.e. Weapon(z))



Now, by propagating substitutions we obtain:



The terminal nodes build the following production:
Depressive(a) & Buy(a, z) & Weapon(z) ⊃ Suicide(a)

BIBLIOGRAPHIE

Textbooks

Dietrich T., London R., Clarkson K. and Dromey G. Appears as chapter XIV in Cohen P. and Feigenbaum E., *The Handbook of Artificial Intelligence*, Vol. III, 323-512, Los Altos, CA: William Kaufmann, 1982.

Kodratoff Y., *Leçons d'apprentissage symbolique*, CEPADUES, 1987, Toulouse, France.

Michalski R., Carbonell J. and Mitchell T., *Machine Learning: an artificial intelligence approach*, Vol. 1, Palo Alto: Tioga, 1983.

Michalski R., Carbonell J. and Mitchell T., *Machine Learning: an artificial intelligence approach*, Vol. 2, Los Altos: Morgan Kaufmann, 1986.

Michalski R., Carbonell J. and Mitchell T., *Machine Learning: A Guide to Current Research*, Boston, Kluwer Academic Press, 1986.

Review

Machine Learning A quaterly journal published by Kluwer Academic Publishers

Buiding Knowledge Bases from Data

[Ganascia 1987] Ganascia J.-G., Learning with Hilbert cubes, in *Proceedings EWSL*, May 1987, Bled, Yougoslavie.

[Hayes-Roth & McDermott 1978] Hayes-Roth F., McDermott J., An interference Matching Technique for inducing Abstractions, CACM, pp. 401-411, 1978.

[Kodratoff & Ganascia 1986] Kodratoff Y., Ganascia J.G., Improving the generalization step in Learning, in *Machine Learning: An Artificial Intelligence Approach, Volume II*, R.S. Michalski, J. G. Carbonell, T. M. Mitchell (Eds.), Morgan Kaufmann 1986, pp 215-244.

[Michalski 1983] Michalski R.S., A Theory and Methodology of Inductive Learning, in *Machine Learning: An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell, T.M. Mitchell (Eds.), Morgan Kaufmann 1983, pp 83-134.

[Michalski 80] Michalski R. and Chilausky R., Learning by being told and learning from examples: an experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis, *Policy Analysis and information systems*, 4 (2), June 1980.

[Mitchell 1978] Mitchell T. Version Space: An approach to concept learning, PhD Thesis, Stanford University, 1978.

[Mitchell 1982] Mitchell T. Generalization as Search, *Artificial Intelligence* 18, 1982, 203-226.

[Mitchell 1983] Mitchell T. Learning and problem solving, in *Proceedings of IJCAI-83*, Karlsruhe, West Germany, 1139-1151, 1983.

[Plotkin 1970] Plotkin G., A note on inductive generalization, in *Machine intelligence* 5, pp. 153-163, eds. Meltzer B. & Michie D., Edinburgh University Press, 1970.

[Quinlan 1983] Quinlan J.R., Learning Efficient Classification Procedures and their Application to Chess End Games, in *Machine Learning: An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell, T.M. Mitchell (Eds.), Morgan Kaufmann 1983, pp 463-482.

[Quinlan 1986] Quinlan J.R. Induction of Decision Tress, *Machine Learning*, 1, 1986, 81-106.

[Vere 80] Vere S. A., Multilevel counterfactuals for generalizations of relational concepts and productions, *Artificial Intelligence*, 14, pp. 139-164, 1980.

Performance Improvement

[DeJong 86] DeJong G. and Mooney R. Explanation-based learning: An alternative view, *Machine Learning*, 1986.

[Fikes 72] Fikes R., Hart P. and Nilsson N. Learning and executing generalized robot plans, *Artificial Intelligence*, 1972, 3, 251-288.

[Laird 87] Laird J., Newell and Rosenbloom P. SOAR: An architecture for general intelligence *Artificial Intelligence*, 1987, 33, 1-64.

[Mitchell 86] Mitchell T. Keller R. and Kedar-Cabelli S., Explanation-based learning: a unifying view, *Machine Learning*, 1(1), 47-80, 1986.

[Mitchell 85] Mitchell T., Mahadevan S. and Steinberg L. LEAP: A learning apprentice for VLSI design. In *proceedings of IJCAI-85*, Los Altos, CA: Morgan Kaufmann, 1986.

ISOLDE: A System for Learning Organic Chemistry through Induction

J. Royce Rose and Herbert Gelernter
Department of Computer Science
SUNY at Stony Brook, NY 11794

1 Introduction

The primary goal of the organic chemistry inductive learning project is the automation of organic chemistry knowledge acquisition. This research program is an offshoot of the SYNCHEM project [3], which has produced a large organic chemistry expert system with a substantive knowledge base of approximately 1000 multilevel frames. With this knowledge base SYNCHEM is able to solve difficult and interesting problems in a circumscribed area of organic chemistry. The current knowledge base represents fifteen years of knowledge acquisition and refinement.

The performance of a large expert system with a nontrivial domain is to a great extent determined by the expertise that is contained in its knowledge base. It is commonly accepted that the acquisition of knowledge from human experts by knowledge engineers is both time-consuming and error-prone. The authors' own experience with expert systems attests to this fact [3] [5] [6] [9]. The ability to automate the knowledge acquisition process would go a long way in overcoming this bottleneck and in increasing the performance, usefulness and long term viability of expert systems as solutions to real-world problems. The existence of several major collections of reaction instances in computer-readable form will facilitate this goal in the domain of organic chemistry.

The organic chemistry inductive learning project comprises three mutually dependent learning problems. The first of the three is the problem of finding initial embeddings of reactant molecules into product molecules. In other words, we must first discover the one-to-one mapping from atoms in the reactant molecules to atoms in the product molecules. The second learning problem is that of extracting a generalized description (reaction schema) for a given reaction class from a selected set of examples. The generalized description is in fact a characteristic description in that it distinguishes a given reaction class from all other reaction classes. The third learning problem is the task of determining taxonomic descriptions. In particular, this is the problem of partitioning a set of reaction instances into reaction classes. Progress has been made on both of the first two problems; the third problem has yet to be addressed. This paper describes the approach employed in finding generalized reaction descriptions and the results that were obtained. Since the approach that was used is primarily based on graph theoretic principles, the results presented in this paper may be expected to find a wider application in other problem domains where graph theory provides the basis for the problem representation.

2 Incremental Learning

A fundamental requirement for a practical inductive learning system in most nontrivial domains is that the incomplete generalizations which are produced be usable, and that these immature generalizations be subject to incremental refinement without the need to reexamine all previous training instances. In complex domains which are still expanding, like organic chemistry, there may be concepts for which it is not always possible to provide enough training instances in order to ensure complete generalizations. It may be the case that not enough training instances are known; it may even be the case that human experts do not yet have a complete understanding of the generalization and are not able to select the appropriate training instances. However these failings have not deterred organic chemists. They are still

able to function productively with incomplete knowledge and through experience, to expand and refine that knowledge.

2.1 The Derivation and Use of Heuristics

In the traditional approach to using heuristics in expert systems, the heuristics are initially perceived and elucidated by the domain expert. The expert system developer then implements the heuristics, usually as part of the expert system code. Machine learning systems can be viewed as expert systems whose area of expertise is learning, and this approach to the discovery and use of heuristics can be viewed as a form of teacher-directed learning in which the heuristics for learning are *coded* into the system. This learning paradigm is not unlike learning by attending lectures.

An important observation is that heuristics are often generalizations based on empirical knowledge. This suggests that heuristics can also be *learned* autodidactically, and gives rise to another approach to the genesis and use of heuristics which is employed by ISOLDE, namely bootstrapping. In this approach, the heuristics are contained in a knowledge base which is maintained by the learning system. As the learning system processes new information, its knowledge base and hence its complement of hypothesized heuristics reflects this additional information. With new information, the system creates new hypotheses, refines some existing hypotheses, and discards others. The critical idea which differentiates this learning approach from many others is that the learning of a new concept does not take place in a vacuum, but is in fact influenced by what the system thinks it already knows. Thus, the more the system *knows*, the easier it is for it to interpret new experiences and add to its knowledge in an intelligent fashion because its inventory of heuristics is more refined and expansive.

2.2 Incremental Knowledge Base

The ability to learn incrementally without the need to reexamine previous training instances requires the ability to *remember* the relevant information contained in previous training instances. Although it is literally true that the previous training instances are not reexamined in their original form, the information they represent must be considered when modifying a generalization. Since it is not known *a priori* which aspects of a training instance are relevant or how many training instances must be remembered, the memory strategy must be capable of storing an arbitrary number of related concepts which may be contained in the training instances the system experiences during its lifetime. The description of this *memory* is a general description of an incremental knowledge base.

2.3 Organization of the ISOLDE Incremental Knowledge Base

2.3.1 Structural Descriptions

The knowledge base contains both observed structural features and hypothesized structural features. The structural features of reactions are organized in a partial order using the *substructure-of* relation. This organization has several advantages. First, and most importantly, the search for related structures is simplified by the fact that structurally similar elements are close in the partial order. In particular, a most specific generalization of a structure (*a largest substructure*) in the knowledge base, if it exists, is an immediate descendent in the partial order. Likewise, a most general structure which is more specific than the structure in question (*a smallest structure containing the structure in question*), if it exists in the knowledge base, is an immediate ancestor in the partial order. In addition, augmenting the knowledge base with new structures is simplified; it is only necessary to determine where the new structure belongs in the partial order.

2.3.2 Attribute and Environmental Descriptions

SYNCHEM uses a backward chaining approach to move from goal molecule to subgoal molecules in its search for ways of synthesizing the target compound. It performs various tests in order to determine what part of its knowledge is applicable at each juncture in the retrosynthesis process. For example, tests are performed to check for certain chemical substructures that might be inadvertently changed by the conditions required for a proposed reaction to take place. A complete description of a reaction must therefore reflect the conditions under which the reaction takes place and must indicate what kinds of structural or functional entities cannot be present in those parts of the target molecule that must remain unaltered by the reaction. This is accomplished by augmenting the structural representation with unary descriptors in much the same way that generalizations are represented by the INDUCE 1.2 system of Michalski [1] [4]. For example, a test instance $p1$ which has an acid sensitive substructure would have the unary descriptor *[acid_sensitive(p1) = TRUE]*. Thus, if it is known that the reaction being generalized proceeds in an acidic environment, (*i.e., (acid_sensitive(r) = FALSE)*), then $p1$ is probably a negative instance. This provides an approach for dealing with negative test instances that is easy to understand and implement, and in which the structural transformation is superficially identical to the structural transformation occurring in positive test instances.

Unfortunately, there is a paucity of attributes associated with individual reactions in the REACS database, (the organic chemistry database ISOLDE is presently using as a source of reaction instances). Also, those provided appear to be selected and described in an imprecise manner, even differing from instance to instance in the same class of reactions. The REACS database is not exceptional in this respect; it is certainly the case that only the most pertinent attributes, if any, are provided with reaction instances appearing in journals. Why are attributes presented in an apparently cavalier fashion in organic chemistry? Quite simply, they exist *ex post facto*. They are a consequence of the structure. Thus, it is expected that a trained chemist can deduce most relevant attributes by analyzing the structure. Since much of the structure is lost in the generalization process, attributes for individual instances must be deduced and later generalized separately but not independently with respect to the structure. The structural and attribute spaces should not be searched independently since it may not be possible to support maximally specific generalizations in both domains [1].

ISOLDE currently deduces acid, base, water and heat sensitivity using the following deduction rule:

$$\frac{P(s_1) \quad (s_1 \subseteq s_3) \quad \neg\exists s_2 (s_1 \subset s_2 \subseteq s_3 \quad s_2 \in F)}{P(s_3)}$$

$P \in \{acid_sensitive \quad base_sensitive \quad water_sensitive \quad heat_sensitive\}$
$F \equiv set\ of\ attribute\ substructures$

This deductive rule asserts that a structure s_3 inherits any acid, base, water, or heat sensitivity possessed by a substructure s_1 unless s_1 is contained in an attribute substructure s_2 which is itself contained in s_3, (in which case s_3 may or may not inherit the sensitivity). In other words, s_3 inherits acid, base, water and heat sensitivity from maximal attribute substructures or substructures containing maximal attribute substructures. This rule does not address the question of *degree* of sensitivity. A refinement of the rule might refer to a set of attribute substructures partially ordered with respect to sensitivity in order to deduce the degree of inherited sensitivity. Additional rules can be added later for deducing electronic, regiospecific and steric properties. Of necessity, these rules will be more complex since inheritance of these properties is not straightforward in that interaction between substructures must be taken into consideration.

2.3.3 Functional Descriptions

In addition to structural and attribute descriptions, descriptions of *functionality* will be required. This is not to be confused with the concept of functional group in chemistry. What is meant in this instance is that an object is assigned to a class if it functions in a manner similar to other members of that class. For example, an object is a seat if it functions as other members in the class of objects determined to be seats. This is in stark contrast to structural classification where an object may be a member of a class only if it structurally resembles other members of the class.

The distinction is of great importance. In some reactions, for example, it may be the case that an activating group must be present at some site in order for the reaction to proceed. Thus two different instances of this reaction might employ different structures that function as activating groups. Generalization limited strictly to structural descriptions will determine that the activating structure is not necessary since in the two cases these structures differ. This will result in flawed generalizations which most likely will not be easily detected. On the other hand, generalization which includes functional descriptions could discover that the presence of an activating group is required for the reaction to proceed.

2.3.4 Alternative Hypotheses

It is often the case that observed data can be explained by more than one hypothesis. Thus the learning system must maintain, re-evaluate, and if necessary reformulate competing hypotheses[2]. There are several types of competing hypotheses that the system must maintain, including reaction generalizations and functional equivalence classes.

The goal of the learning system is to induce reaction generalizations. Consequently, for each reaction the system is trying to learn, it maintains competing reaction generalizations until it has been presented with enough information to converge to a single generalization.

The system will also hypothesize functional equivalence classes in order to discover classes of structures which are functionally equivalent but structurally dissimilar. It is expected that domain knowledge will be useful in constraining the number and content of hypothesized equivalences. While hypothesized reaction generalizations are expected to apply to a single class of reaction instances, the class defined by the generalization, hypothesized equivalence classes are expected to apply universally. In other words, if a member of a functional equivalence class appears in a reaction instance, then substituting any other member of its class for that member should in general not affect the reaction.

3 Inductive Generalization of Reaction Instances

3.1 Descriptive Generalization strategies

Descriptive generalization strategies can be divided into three broad classes: data-driven, model-driven and statistical pattern recognition strategies [1] [7]. For these types of strategies, hypothesized generalizations of descriptions are accepted or rejected according to how accurately they classify the test cases presented. Test cases are identified as being either positive or negative instances. A positive instance is a test case which the hypothesized generalization must satisfy. Conversely, negative instances must not be satisfied by hypothesized generalizations.

In statistical pattern recognition strategies, test instances are represented by points in n-space and generalizations are represented by surfaces in n-space. A test instance is classified by a generalization according to whether its corresponding point in n-space lies to one side or the other of the decision surface corresponding to the generalization. Although the field of Statistical Pattern Recognition has developed very good generalization methods for certain classes of decision surfaces, this method is not as suitable for finding generalizations of reaction instances as are data-driven or model-driven strategies because of the difficulty in defining a suitable n-space in the absence of a complete set of test instances at the outset.

The generalization process can be viewed as a search through the hypothesis space for adequate generalizations. The generalization language defines the possible hypothesis space. A key characteristic of the problem of generalization as search is that the search space is partially ordered by the *more-specific-than* relation [7] [8]. This underlying structure of the search space is fundamental to the approach, since a generalization strategy must be able to compare a hypothesized generalization with a test case in order to determine whether it is more general or less general than the test case.

The model-driven and data-driven strategies differ primarily in the manner in which new hypotheses are formed. In the case of model-driven strategies, new hypotheses are based on a model of the description space, independent of the test cases. The test cases are then used to weed out unsatisfactory generalizations in order to find a few *best* hypotheses which satisfy the test cases. The model-driven strategy is often referred to as the generate-and-test, or top-down method. Data-driven methods, on the other hand, generate hypotheses either directly or indirectly from the test cases and do not require a model of the description space. The data-driven strategy is often referred to as a bottom-up method.

Data-driven strategies use the differences between hypothesized generalizations and the input test cases to guide the search process through the hypothesis space. Thus, data-driven strategies reformulate generalizations in direct response to observed discrepancies between the generalizations and the test data. In general, hypothesized descriptions are made less restrictive by dropping conditions or descriptive clauses in order to match positive test instances. Conversely, hypothesized descriptions are made more restrictive by adding conditions or descriptive clauses in order to exclude negative test instances. As a result, data-driven methods are significantly more sensitive to *noise* in the input data than are model-driven methods. Such strategies may not be appropriate in domains where the test data are not reliable.

3.1.1 Version Space

One data-driven strategy which is particularly well suited to finding a generalized description for a reaction class from instances of that class is the version space approach [7] [8]. With this generalization method, the set of all possible hypotheses which are consistent with all observed test cases is revised with each new training instance. This set of hypothesized generalizations constitutes the *version space*; it is reduced during the search process. Referring to the set of all possible hypotheses as the version space is appropriate since it contains all possible versions of the emerging generalization.

Exploiting the partial ordering of the hypothesis space, the version space is represented by two sets of generalizations, S and G, which constitute specificity and generality bounds, respectively, on the version space. Mitchell [7] defines the set S as follows:

$S = \{s \mid s$ is a generalization that is consistent with the observed instances and there is no generalization which is both more specific than s and consistent with the observed instances.$\}$

Thus the definition of S requires that all generalizations contained in it must be maximally specific and consistent with all observed test instances. In addition to the set S, the version space strategy maintains a set G which forms a generality bound on the version space. The set G is defined by Mitchell as follows:

$G = \{g \mid g$ is consistent with all observed instances, and there is no generalization which is both more general than g, and consistent with the observed instances.$\}$

Thus the generalizations contained in G are maximally general and consistent with all observed test cases, both positive and negative.

From the definitions of S and G, it is clear that one of the key requirements of the generalization language used in a version space approach is the ability to support comparison of generalizations in order to determine whether one generalization is more general than another. As a practical matter, it is important that a direct comparison be possible without having to resort to comparing the sets of test cases that the generalizations represent.

Since the sets S and G delimit the version space, it is possible to determine whether or not a given generalization lies in the version space. A generalization lies in the version space iff it is more specific or equal to some member of G and is less specific or equal to some member of S.

The use of a test case as a training instance constitutes a step in the search. Both sets S and G are revised in order to maintain consistency with the new test case. The set S is computed by a breadth-first search from very specific hypotheses to more general hypotheses. If the test case is a positive instance then all generalizations in S which fail to match it are made less restrictive in order to match it. Those resulting generalizations which are overly general according to the definition of S are discarded. If the test case is a negative instance then all generalizations in S which match it are considered overly general and are discarded. By definition of S, there are no generalizations more specific which are consistent with the observed test cases; therefore those generalizations which match the negative instance can not be made more specific and must be discarded. Thus the breadth-first search by which S is computed proceeds monotonically in the partially ordered search space from specific to general descriptions.

The set G is computed by performing a breadth-first search in the opposite direction. In other words, the breadth-first search proceeds from general hypotheses to more specific hypotheses. This is possible since the partially ordered hypothesis space can be searched in either direction. If the training instance is a positive instance and G contains generalizations which do not match it then these generalizations are discarded since they are overly specific. If the training instance is a negative instance then generalizations in G which match it are made more specific so that they no longer match it. By definition of G, these generalizations are only made as specific as necessary so that they remain consistent with observed test instances. The generalizations in G may become more specific but never more general, since G is the generality bound on the version space.

It should be noted that when a generalization in set S or G is modified it need only be compared with the generalizations in sets S and G, and not with any of the observed instances in order to assure consistency, since G and S delimit the version space. Thus S summarizes the information from the positive training instances and G summarizes the information from the negative training instances. In addition to the requirement that the generalization language support a *more-specific-than* relation which imposes a partial ordering on the hypothesis space, every chain in the partially ordered space must have a least upper bound and a greatest lower bound. This additional requirement is necessary so that S and G can delimit any possible version space that may arise[7].

The sets S and G also provide a means of knowing when the generalization process is complete. This occurs when the sets S and G reduce to the same set. Even if S and G do not converge to the same set with the available training instances, they provide the ability to use incomplete generalizations. Since they delimit the version space, they can correctly identify instances that lie outside of the version space as positive or negative instances. In addition, they can be used to guide the search by suggesting training instances which are not equal to any of the generalizations contained in S or G but which lie within the version space. Since training instances which do not lie in the version space cannot contribute any new information, they may be discarded. It may also be possible to use the sets G and S to fabricate new instances and, with the help of a teacher to classify the resulting cases as negative or positive, to reduce the version space.

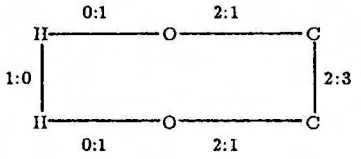
3.2 The Generalization Language

Organic reactions are traditionally represented by pairs of graphs. The graphical representation has a left hand side and a right hand side corresponding to *before* and *after* the reaction has occurred, respectively.

Example:

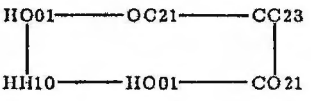
$$H_2 + O = C = C = O \longrightarrow HO - C \equiv C - OH$$

If the reaction were described by a single graph, the comparisons between reaction instances would be simplified. For example, the above reaction could be represented as a single graph with bonds that vary in time [10]. The following graph indicates that the double bond connecting the two carbon atoms changes to a triple bond and that the double bonds connecting the oxygen atoms to carbon atoms are replaced by single bonds.



Using this representation, bonds are labeled with their *before* and *after* values. If a new single bond is created in the reaction where none previously existed, it is labeled 0:1. Likewise, a single bond which is broken would be labelled 1:0. Both of the presentations above are described as *atom-centered.*

In this work, all atoms participating in the reaction, both before and after the transformation, will be represented explicitly. Since only the bonds change under this scheme and not the atoms, a *bond-centered* presentation is preferred. In the bond-centered representation, a reaction is described by a single connected labeled graph in which the time-varying bonds constitute the labeled nodes [10]. The example above can be described by the following bond-centered graph:



Note that the node label $OC21$ is equivalent to $CO21$, since both indicate a double bond between an oxygen atom and a carbon atom changing to a single bond. However, the node label $OC21$ is not equivalent to $OC12$ since in the first instance a double bond becomes a single bond while in the second instance the reverse occurs.

As mentioned earlier, the generalization language defines the hypothesis space. It determines what kinds of generalizations are possible. It also greatly influences the choice of generalization strategies. The bond-centered presentation provides a concise, easy to use representation which simplifies the comparison of reaction instances. It also makes it easy to organize reaction instances and generalizations in a partial order using the *more-specific-than* relation. If graph b is a proper subgraph of graph a then a is more specific than b.

In addition to structural descriptions, ISOLDE's generalization language must also support attribute and functional descriptions. The language of bond-centered graphs extended with unary descriptors and equivalence sets provides this ability. The use of equivalence sets offers a simple way of expressing structurally dissimilar but functionally equivalent structures.

3.3 A Modified Augmented Version Space Approach

The organic chemistry inductive learning problem is well suited to a modified augmented version space approach in which descriptive and functional generalizations are refined simultaneously, although separately. The use of bond-centered graphs as a means of representing both test cases and generalizations supports the *more-specific-than* relation and provides the required underlying hypothesis space structure essential to the dual breadth-first search process. The structural and attribute spaces are searched separately but not independently. As functionality is represented by equivalence sets, functional generalization occurs in structure

space. Since the attribute space described by unary descriptors is not rich enough to distinguish all reactions, a version space approach is not possible here. A simpler breadth-first strategy is employed for searching the attribute space.

3.3.1 Attribute Space

The attributes which ISOLDE deduces denote tolerance to acid, base, water and heat sensitivity in reaction instances. From the deduced attributes ISOLDE generalizes both positive and negative sensitivity. For example, if some of the reaction instances contain acid and/or water sensitive maximal attribute substructures and none contain heat or base sensitive maximal attribute substructures then the attributes ISOLDE hypothesizes for the reaction are: *(can_have:: acid_sensitivity water_sensitivity)* and *(probably_cannot_have:: heat_sensitivity base_sensitivity).*

The four attributes deduced and the related attributes which ISOLDE consequently generalizes are by themselves not sufficient for uniquely describing reactions. Thus it is not possible to distinguish among some reaction classes solely on the basis of this limited set of attributes. Recalling that the set G in version space represents the information contained in negative instances, or in other words, instances of reactions different from that class being generalized, it becomes apparent that the set G cannot be supported without a richer set of attributes. Consequently, no attempt has been made to develop the set G in this limited attribute space, nor are negative instances used to prune hypotheses in the set S. This initial strategy may be viewed as a semi-version space approach proceeding from restrictive to less restrictive descriptions. In this case the most specific hypothesis is the most restrictive: *(cannot_have:: acid_sensitivity base_sensitivity water_sensitivity heat_sensitivity).* The most general hypothesis is the least restrictive: *(can_have:: acid_sensitivity base_sensitivity water_sensitivity heat_sensitivity).*

3.3.2 Structural Version Space

Domain-specific knowledge can be used in determining how to initialize the set G, which represents the generality threshold of the structural version space. Since the breadth-first search used to update G traverses the hypothesis space from general to more specific structural descriptions, G should initially contain the most general plausible description of the reaction structure. It is known that all instances of a reaction must share the same set of active bonds (*in the case of bond-centered graphs, these are the bonds whose multiplicity changes during the reaction*). Thus, at any given instant, all hypotheses in G must contain this information. The most general structural description would therefore be the smallest connected bond-centered subgraph of a positive instance containing all active bonds in which intervening bonds which are not active are replaced by variable nodes (*nodes which may be replaced by arbitrary subgraphs*). Following this reasoning based on problem-specific knowledge, G can be initialized with the smallest connected bond-centered subgraph containing all active bonds from the first positive test case.

In contrast to G, the set S represents the specificity threshold of the structural version space. The breadth-first search used to update S traverses the hypothesis space from specific to more general structural descriptions. Thus S should initially contain the most specific known structural description, which is trivially the first positive test instance.

The development of the descriptions contained in set S is straightforward. As more positive reaction instances are examined, the descriptions contained in S are made more and more general (*i.e.*, the graphs become smaller and smaller) so that the observed differences between the hypothesized generalizations and the observed test cases are reconciled.

In order to refine the set of descriptions contained in G using only the *more-specific-than* relation, it will be necessary to have known negative test instances which have active bond information identical to the positive test instances. As discussed in the section describing the concept of version space, G represents the structural information contained in the negative

test instances. By initializing G to contain the smallest connected subgraph incorporating the active bonds, G represents the information contained in negative test instances which have active bond information which differs from the positive test instances. Therefore, only the information contained in the negative instances whose active bonds are identical to the positive instances is missing from G.

4 Preliminary Results

The following results were produced by the initial implementation of ISOLDE in which the generalization process is limited to the search of structure and attribute space. Thus, functional descriptions, which are represented by equivalence sets, are not included in the examples. Although the final generalization of the specificity threshold S contains only a single element in the following cases, note that intermediate generalizations of S sometimes contained more than one hypothesized generalization. It should also be noted that the attributes deduced and generalized by ISOLDE are by no means exhaustive and form a subset of those considered by SYNCHEM. In addition, some attribute substructures may belong to more than one class and consequently may be in a *cannot_have* list for any of several reasons. Thus, it is to be expected that there be some disagreement between ISOLDE's results and SYNCHEM's knowledge base. The reaction instances used in this exercise are taken from the REACS database and were provided by the Eastman Kodak Company.

4.1 Example: Wittig Reaction

ISOLDE was presented with two sets of examples of actual reaction instances representing two variations of the Wittig reaction. The first set contained seven positive cases which were sufficiently different so that ISOLDE was able to generalize the initial set S, representing the specificity threshold of the structure version space, considerably (see figures 1 and 3). Since no negative examples were available, the set G, representing the generality threshold, could not be made more specific (figure 2). The results of generalization of the attribute space are: *(can_have:: base_sensitivity acid_sensitivity), (probably_cannot_have:: water_sensitivity heat_sensitivity).* The attributes in the *probably_cannot_have* did not appear in any of the examples. The *can_have* list is consistent with SYNCHEM's knowledge base, which was built in the traditional manner by domain experts. The primary difference is that SYNCHEM specifies these attributes in greater detail (*i.e.*, indicating which particular attribute substructures are or are not allowed). The *probably_cannot_have* list, which denotes the sensitivities not observed, does not quite agree with SYNCHEM's knowledge base. Of the fourteen water sensitive attribute substructures known to SYNCHEM, eight are in its *cannot_have* list for the Wittig reaction, while the remaining eight are allowed. Seventeen of the the thirty-two heat sensitive attribute substructures known to SYNCHEM are in its *cannot_have* list for the Wittig reaction.

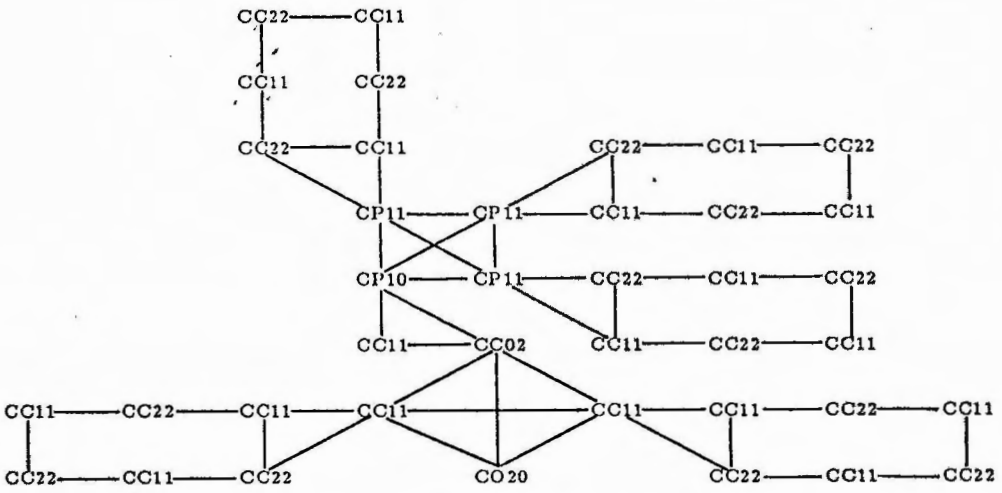


Figure 1: The initial specificity threshold S_0 for first set of Wittig examples

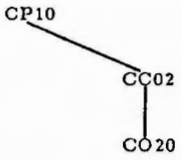


Figure 2: The generality threshold G_0 for first set of Wittig examples

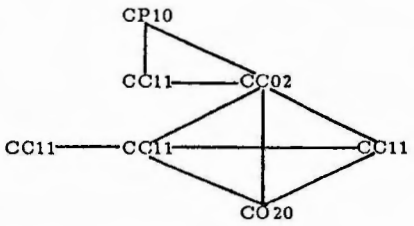


Figure 3: The generalized specificity threshold S_6 for first set of Wittig examples

The set of instances for the alternate version of the Wittig Reaction contained six positive examples which were so similar that very little generalization of the set S was possible (see figures 4 and 6). Given a larger set of examples, such a result would tend to support the hypothesis that the set S is very close to the description of the underlying reaction process. As in the previous case, there were no negative examples with which to make the set G more specific (fig. 5). As one should expect, results identical to the first set of Wittig instances were obtained in the generalization of attributes.

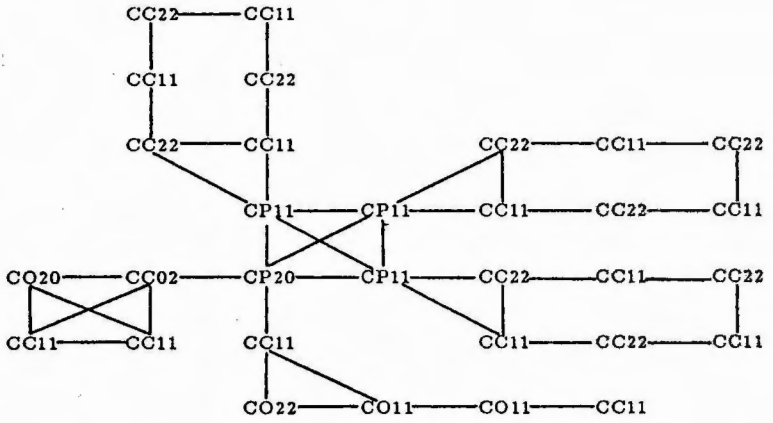


Figure 4: The initial specificity threshold S_0 for second set of Wittig examples

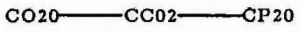


Figure 5: The generality threshold G_0 for second set of Wittig examples

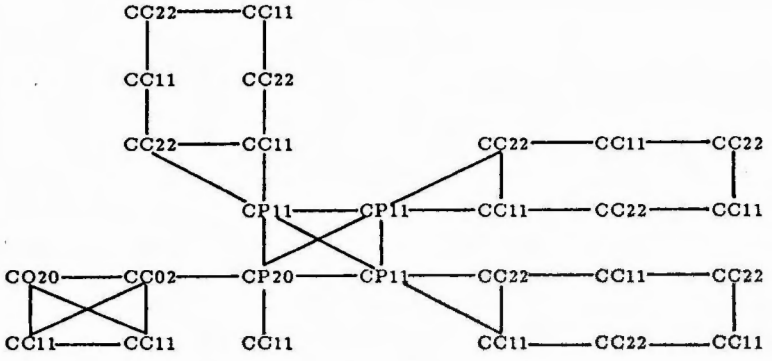


Figure 6: The generalized specificity threshold S_5 for second set of Wittig examples

These two examples illustrate a problem that haunts organic chemistry and many other fields of science; the results of *unsuccessful* experiments are rarely published. Thus scientists are condemned to repeat the failures of their predecessors. The unfortunate result is that the knowledge which is the fruit of unsuccessful efforts is lost and there is a dearth of negative examples for use in inductive learning experiments. When used interactively, a system such as ISOLDE, which can exploit and store the essence of negative examples, serves an additional valuable role as a repository for such knowledge which would otherwise be lost.

4.2 Example: Diels-Alder Reaction

The Diels-Alder example illustrates the importance of functional generalization and the inadequacy of a purely structural approach in the domain of organic chemistry. The starting sets S_0 and G_0 representing the specificity and generality thresholds are shown in figures 7 and 8, respectively. The initial implementation of ISOLDE does not attempt functional

generalization; consequently the presence of functionally equivalent but structurally dissimilar structures (figure 10) in the set of Diels-Alder reaction instances remains undetected. This results in the over-generalization of set S (figure 9). The results of the generalization in the attribute space are: *(can_have:: acid_sensitivity base_sensitivity water_sensitivity heat_sensitivity)*. These results are consistent with SYNCHEM's knowledge base.

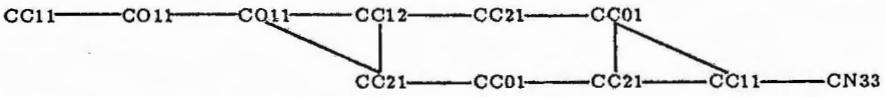


Figure 7: The initial specificity threshold S_0 of the Diels-Alder example

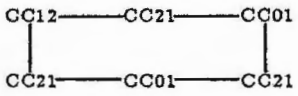


Figure 8: The generality threshold G_0 of the Diels-Alder example

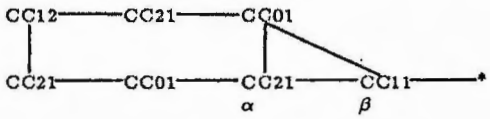


Figure 9: Over-generalized specificity threshold S_n. An activating group required for the reaction to occur is missing at the asterisk and at node β

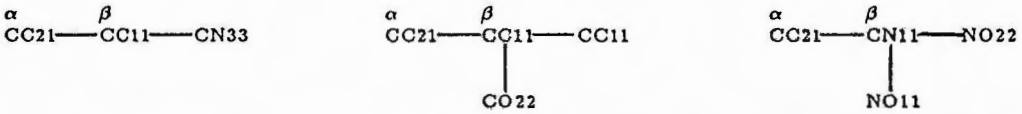


Figure 10: Functionally equivalent but structurally dissimilar structures containing activating groups. The functionality provided by these structures is required in order for the Diels-Alder reaction to occur. The nodes labelled α and β correspond to the similarly labelled nodes in figure 9

4.3 Example: Pinacol Rearrangement Reaction

The Pinacol Rearrangement example illustrates the importance of having a representative set of examples for concept learning. If the examples are too similar and do not reflect the diversity of instances in which a concept may appear, then one incorrectly generalizes that all instances are practically uniform, regardless of the number of cases. Conversely, if the examples reflect the multiformity of the concept, then it is often possible to learn a concept with a small number of examples. In the case of the Pinacol Rearrangement reaction, the set of four examples are sufficiently representative of the structural concept that ISOLDE learns the same structural concept as contained in SYNCHEM's knowledge base. In contrast, the four examples are not indicative of the diversity of attribute substructures which may occur. The attributes ISOLDE hypothesizes are: *(probably_can_have:: acid_sensitivity)*, *(probably_cannot_have:: base_sensitivity water_ sensitivity heat_sensitivity)*. This does not agree with SYNCHEM's knowledge base which indicates that several attribute substructures from each class of sensitivity are allowed.

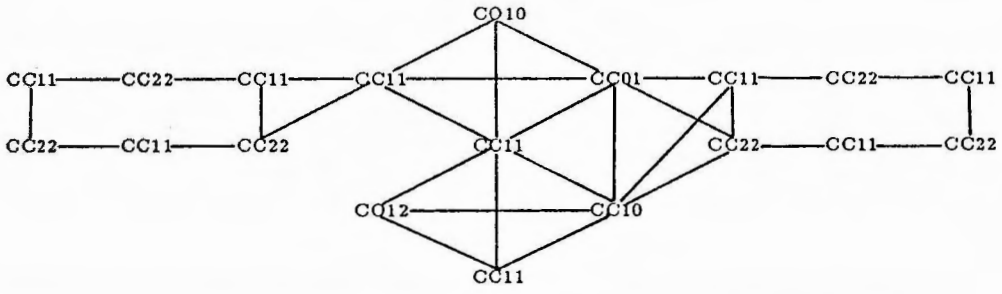


Figure 11: The initial specificity threshold S_0 for the Pinacol Rearrangement example

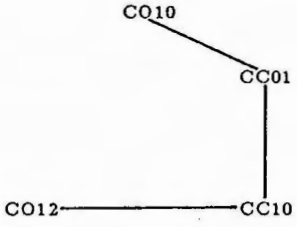


Figure 12: The generality threshold G_0 for the Pinacol Rearrangement example

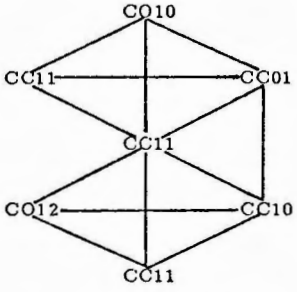


Figure 13: The generalized specificity threshold S_3 for the Pinacol Rearrangement example

5 Summary

This paper describes preliminary work on a system for inducing generalized descriptions of synthetic organic reactions. The system uses an incremental knowledge base to *remember* its learning experiences and to maintain and rank competing hypotheses. As in other fields of science, the knowledge gained from *unsuccessful* experiments is not generally available to the community of organic chemists as a whole, since it is rarely published. The system's incremental knowledge base can serve the additional important role of being a repository for the knowledge contained in negative examples of reactions, thus making this important knowledge more widely available.

The system, while presently incomplete as discussed in the results section, provides the basis for inducing complex descriptions which describe the structural, environmental and functional aspects of organic chemical reactions. In addition, the system offers many tools which will prove useful for extracting taxonomic descriptions of reaction instances.

We gratefully acknowledge the support of the Eastman Kodak Company in providing both major funding for this project and access to portions of its computer-readable database of synthetic organic reactions. Additional support for the development of the SYNCHEM system testbed for the pursuit of this research was provided by National Aeronautics & Space Administration Grant No. NAG3651.

References

[1] T. G. Dietterich and R. S. Michalski. Inductive learning of structural descritpions. *Artificial Intelligence*, (16):257–294, 1981.

[2] W. Emde. Non-cumulative learning in metaxa.3. In *Proceedings of the 10th Internat. Joint Conf. on Artificial Intelligence*, pages 208–210, 1987.

[3] H. Gelernter, G. A. Miller, D. L. Larsen, and D. J. Berndt. Realization of a large expert problem-solving system. In *IEEE 1984 Proceedings of the First Conf. on Artificial Intelligence Applications*, pages 92–106, 1984.

[4] Ryszard S. Michalski. *Machine Learning*, chapter A Theory and Methodology of Inductive Learning. Tioga Press, 1983.

[5] P. L. Miller, C. Shaw, J. R. Rose, and H. A. Swett. Critiquing the process of radiologic differential diagnosis. *Computer Methods and Programs in Biomedicine*, (22):21–25, 1986.

[6] P.L. Miller, S. Blumenfrucht, J.R. Rose, M. Rothschild, H.A. Swett, G. Weltin, and N. Marrs. *Selected Topics in Medical Artificial Intelligence*, chapter HYDRA: A Knowledge Acquisition and Verification System. Springer Verlag, 1988.

[7] T. M. Mitchell. Generalization as search. *Artificial Intelligence*, (18):203–226, 1982.

[8] T. M. Mitchell. Version spaces: a candidate elimination approach to rule learning. In *Proceedings of the 5th Internat. Joint Conf. on Artificial Intelligence*, pages 305–310, 1977.

[9] H. A. Swett, C. Shaw, J. R. Rose, and P. L. Miller. Icon: an expert system for critiquing radiologic differential diagnosis. In *Proceedings of the American Assoc. for Med. Systems and Informatics*, pages 202–206, 1985. Congress-85.

[10] C. S. Wilcox and R. A. Levinson. A self-organized knowledge base for recall, design and discovery in organic chemistry. In T. H. Pearce and B. A. Holme, editors, *AI Applications in Chemistry*, American Chemical Society, 1986. ACS Symp. Series 306.

Towards robust expert systems that learn from experience
- an architectural framework

Agnar Aamodt

Knowledge Engineering Laboratory, ELAB-RUNIT, SINTEF
N-7034 Trondheim-NTH, Norway
and
Department of Electrical Engineering and Computer Science
University of Trondheim
aamodt%vax.runit.unit.uninett@tor.nta.no

Abstract
This work focuses on two major weaknesses of current knowledge-based systems: Their brittle competence and their inability to learn from experience. As a step towards attacking these important and highly interconnected problems, a KBS architecture that integrates case based reasoning and learning into a more general knowledge acquisition and problem solving framework is proposed. The importance of developing a conceptual knowledge model of the domain - tightly interconnecting concept-descriptions, relations, constraints, etc. is emphasised. This knowledge, as well as knowledge in the form of heuristic rules and previously solved cases are defined within the same knowledge structure, making it one unified knowledge base. The conceptual model should look into bordering domains of the actual application, as well as parts of the general world related to the domain. In this way, more robustness in problem solving and better support for the case based learning process is achieved. We are developing a system - called CREEK - to support the building and continuous updating of diagnosis type expert systems along these lines. A top-level control strategy monitors the problem solving process, and decides whether the conceptual knowledge model, past cases or heuristic rules should be activated in the attempt to achieve the wanted goal. A generic model of how to solve diagnosis problems guides this process. The learning method involved is knowledge intensive, case based learning. Learning becomes a part of the problem solving process, thereby obtaining incremental learning as a natural part of the system's problem solving effort.

I. Introduction

It is generally agreed that more powerful KA tools and techniques are needed in order to increase both the quality and the quantity of knowledge based systems for real world applications. By real world systems in this context, we refer to systems that are supposed to be used daily to solve non-trivial, practical problems in cooperation with people and/or other systems. That is: systems that exhibit a certain level of complexity, that sometimes have to cope with problems on the border of (or slightly outside) their special domain of competence, and that have to be properly updated and maintained in order not to degrade over time.

The motivation behind this research is to attack two major weaknesses of current knowledge based systems, namely their brittle competence and their inability to learn from experience. Some degree of robustness and the ability to learn from solving new problems seem crucial to the notion of real world knowledge-based systems, and actually to any system that exhibit a type of behaviour that we may want to call intelligent. Robustness means being able to cope with unexpected situations in an intelligible and 'graceful' manner, which - compared to current systems - implies improved problem solving performance within the domain, some way to handle problems slightly outside the actual domain, and more intelligent dialogues with users and experts.

People are robust problem solvers since we possess a coherent body of general world knowledge,

specific knowledge of domains - from first principles to surface level heuristics - and a huge library of specific past problem solving cases. This gives us a fundamental understanding of the world we operate in, as well as a memory of past problem-solving episodes integrated into this fundamental knowledge structure. While solving problems, we are frequently reminded of similar past problems. Using more general knowledge as support, we are able to adapt the solution (or solution path) of the retrieved case to solving the new problem. Problem solving strategies help us to focus the use of knowledge within the current context and according to our goals. Through success and failure in achieving our task, we learn to do things better the next time.

In this perspective, robustness in problem solving is dependent on an ability to adapt behaviour according to experience. The properties of robust competence and continuous learning are highly interdependent: By having a robust body of knowledge to support experiential learning, the learning process can be more focused and productive, and the results more easily applicable. To cite Marvin Minsky: "The more you know, the more - and faster - you can learn". (See Lenat-87 for an enlightening discussion on the role of knowledge in machine learning and general AI). And vice versa: Being able to learn from each new experience gradually builds up a more knowledgeable - and therefore more robust - system. Provided, of course, that appropriate methods exist to incorporate the new knowledge into the existing knowledge base. A system that effectively learns each time it attempts to solve a problem will then gradually acquire a high degree of understanding of the world it operates in. This kind of learning can only be obtained if the learning method uses the existing body of knowledge to incrementally update the system's knowledge base. The recent developments within case based reasoning and learning indicate a very promising direction to pursue in order to achieve this behaviour in a knowledge based system.

A robust competent problem solving and learning system needs to have both a fairly broad and a fairly deep understanding of its domain. Our experience is that for most domains this is most effectively done by initially developing a knowledge model as a cooperative effort between an expert and a knowledge engineer. The subsequent learning should use the existing knowledge to the extent that plausible inferences can be drawn (coherent explanations can be generated), but the system should be able to ask the user/expert for confirmation or explanations whenever it is necessary. The knowledge model should describe the relevant specific and general knowledge within a common framework, and include strategic problem solving knowledge as well as object level domain knowledge. It should be deep enough to tightly connect concepts, relations, processes, constraints, rules, etc., and wide enough to incorporate relevant descriptions of neighbouring domains as well as some parts of the general world. The purpose of the model is to give the system a rich knowledge fundament for problem solving and learning. It will support case based learning and reasoning by generating explanations to justify reasoning steps, extract relevant features, retrieve relevant cases, index cases , decide which parts of a case to store and which previous cases to remove.

An increasing body of research is seriously attacking some of the central problems involved. During the last few years a lot of research has focused on analysis and modelling of knowledge, and the importance of developing an explicit knowledge model (expertise model) of the domain - modelling both factual knowledge, strategies, and problem solving methods (Clancey-84, Wielinga-86, McDermott- 86, Keravnou-86, Breuker-87, Steels-87, Sticklen-87, Gruber-88).The most ambitious - and long term perspective - approach is probably the CYC project at MCC (Lenat-86, Shepherd-88). This project aims at building a huge knowledge base capturing essentially all the common sense knowledge a human being has. Upon this core of common sense, more specific systems like natural language understanders, expert systems, machine learning systems, etc. may be built. The common sense knowledge will help to structure and organise the domain knowledge of the application, and give systems a self-understanding that will be unique to this approach. The general world knowledge will also serve as the knowledge to fall back on whenever the more specific knowledge fails to apply, thereby achieving a high degree of robustness.
Within the machine learning community, we have recently seen a shift from similarity-based, knowledge-poor induction methods towards more knowledge intensive learning techniques. Classical induction methods like Version Space (Mitchell-82), AQ (Michalski-83) and ID3 (Quinlan-86) are not suitable for this kind of learning. Besides being unable to take advantage of an evolving knowledge base in their learning process, they also assume a classical view of concept definition. This view defines a concept as a set of attributes that are singly necessary and jointly sufficient to classify examples of the concept. Such definitions may be adequate for mathematical and other strictly formal concepts, but as shown by several authors (Wittgenstein-53, Smith-81, Schank-86a)

they simply do not exist for natural concepts like chair, bird, games, etc. Defining a diagnostic hypothesis (e.g. senility, essential-hypertension) is often subject to the same difficulties. That is, a fault is described by features (e.g. symptoms) that to a larger or lesser degree supports the hypothesis. The degree of support typically depends on the context or perspective in which that actual reasoning takes place. Smith & Medin distinguish between the 'classical', the 'probabilistic' (e.g. degree of prototypicality) and the 'exemplar' (purely extensional) view of concept definitions. Methods that try to learn non-classical concept definitions all need support from domain knowledge in order to explain why an exemplar is within the range of a concept, why some features are more relevant than others, etc. These explanation based learning methods include purely deductive methods based on a complete domain theory (EBL/EBG, see Mitchell-86 and DeJong-86) as well as methods to generate and use plausible explanations in an incomplete knowledge base (Schank-86b, Branting-88, Murray-88). We will also include knowledge intensive case based reasoning/learning (Kolodner- 83a, Kolodner-83b, Carbonell-86, Porter-86) among the explanation based learning methods.

In the next chapter we briefly describe the approach wherein which our framework is defined, illustrated by a life cycle model of the total knowledge acquisition process. In chapter 3 a summary of case based reasoning/learning is given, followed by a discussion of the proposed framework. The application being developed in order to test the framework is sketched in chapter 4, followed by a status summary. Finally, some guidelines for evaluating the framework is listed.

II. An integrated approach to problem solving and learning

Our approach to knowledge acquisition, problem solving and machine learning is:

* **Interactive knowledge acquisition** by developing a thorough knowledge and problem solving model of the domain. This model should be deep enough to tightly connect the concepts, relations, processes, constraints, etc. of the domain. The model should be wide enough to incorporate relevant descriptions of neighbouring domains as well as parts of the general world. We are talking about a much more extensive domain model than the 'list some domain concepts and rules' approach still common in the development of simple expert systems. To achieve some degree of robustness, it is important to give a broader description of the domain than what might seem required by the actual application. Interactive KA tools must be designed for this kind of modelling.

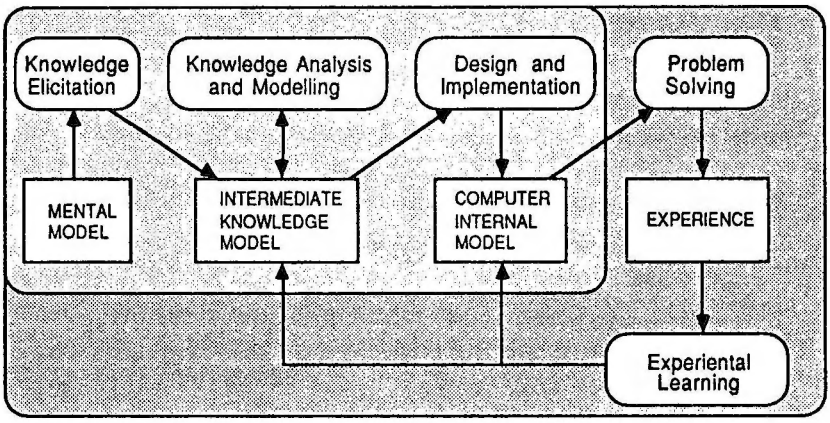
* **Problem solving** by combining case based reasoning with reasoning from more general domain knowledge (concept inheritance hierarchies, heuristic rules). A goal- and task oriented problem solving strategy is required, in order to decide when to use cases, when to use more general knowledge and how to combine them.

* **Machine learning** as an integrated part of the problem solving process, by retaining the problem solving cases and incorporating them into the base of more general knowledge. The knowledge base will be actively used in the process of input feature selection, case retrieval and matching, evaluation of similarities between cases, transforming the solution of a past case to a new problem, and in deciding what to be stored from - and updated on the basis of - the new case. If the knowledge base is not able to generate the appropriate explanations for these tasks, the user is asked for confirmation or supplementary information. This makes the running system a learning apprentice;, observing the user's actions and initiating a dialogue if necessary. If the user is unable to give a reliable answer, the question is saved for later discussions with an expert and/or knowledge engineer.

The figure on next page illustrates our integrated process-and-storage model of knowledge acquisition and problem solving.

The process of **knowledge elicitation** tries to get the relevant knowledge out of the expert, in some form or another. This unstructured knowledge is then gradually refined into an integrated intermediate knowledge model, capturing strategic and tactical problem solving knowledge as well as object level knowledge of the domain.

The goal of the **knowledge analysis and modelling** phase is to come up with a problem solving and knowledge model for the application. This model should be based on characteristics of the domain as such, and not on implementational constraints given by a particular piece of software or hardware. An expressive representation language - which is able to integrate specific problem solving cases - is needed. The study of interactive tools to support this modelling activity is a rapidly increasing area of research. The modelling process constitutes a very important - but often underestimated - phase in interactive knowledge acquisition.



The Knowledge Acquisition Cycle

The rounded boxes are processes, while the sharp-cornered boxes are virtual stores of knowledge and information. Arrows are directed knowledge- and information flow. The focus is on acquisition and maintenance of knowledge and problem solving expertise.

The **design and implementation** phase includes choosing/developing the languages for representing the knowledge and executing the problem solving methods that are parts of the (intermediate) knowledge model. If we look beyond the actual knowledge based part, this phase also includes designing and implementing the total system with its interfaces to people and other systems. A knowledge based system for real world applications is subject to all the general requirements of computer systems in industry and elsewhere. It is extremely important that the AI part is hooked into the general information system model of the application. Since these design and implementation aspects are somewhat beyond the scope of this paper, they will not be further discussed.

Ideally, the implementation and design phase should be separated from the knowledge modelling phase, since the main purpose of knowledge modelling is to understand and structure the domain by describing it in a coherent and implementation-independent way. The modelling representation language (e.g. as used in an automated analysis/modelling tool) should therefore be flexible and highly expressive, while the representation in the target system will have to sacrifice some of these properties for run-time efficiency. This independence in representation is, however, difficult to achieve - particularly if we want some degree of automated transformation between the two knowledge models. The transformation of knowledge between different representations is in general very difficult to fully automate. The kind of compromises to be made between the two representation languages will depend on the particular domain and application, and the degree of automation of the transformation process being achieved.

III. Case based reasoning and learning (CBRL[1])

The idea behind case based learning is different from 'example learning' (as the term is typically used in the machine learning literature). In example learning (learning by examples) a set of examples are used to induce generalised concept definitions. The learned concept classification rules are later used in problem solving, i.e to identify (classify) instances of the concept. In the case based learning approach the cases themselves are used in later problem solving. Past cases are indexed and stored in order to facilitate later retrieval and use in solving new and similar problems. A case typically contains a sequence of features describing the problem, parts of the path towards the solution of the problem, and the solution itself. (Analogical reasoning may be regarded as a kind of case based reasoning, where the similarities between cases are more abstract, and often involving functional and causal relationships across domains.)

Case based learning represents an integrated approach to problem solving and machine learning:

A problem is solved by retrieving a problem case from the past, similar to the new one, and adapting its solution to solving the new problem. Important subtasks typically are:

- select and/or infer relevant features from the problem description
- find a past case that (best) matches the current problem
- decide what parts of the past solution to use or transform in order to solve the new problem
- apply the past solution to the new problem and test the expectations created
- revise solution, select another case or accept solution

The system learns by incorporating each new case into the case base. The tasks involved in this process may include:

- decide which parts of the case to store
- create remindings to the new case from relevant features, to facilitate fast retrieval by future similar cases
- establish difference links in order to distinguish several similar cases
- generalise two or more cases to form 'super cases'
- update the case structure as a result of failure during problem solving (e.g. if the system retrieves a case which leads to an incorrect or unacceptable solution, the strength of some of the indices pointing to that case may have to be re-evaluated)
- forget cases that have not been used over some period of time, thereby deleting irrelevant cases to regain space and improve efficiency

For intelligent and robust behaviour, both the problem solving and the learning processes need to be guided by the existing knowledge in the system. Otherwise we end up with an essentially syntax based method where matching (for example) consists of looking for syntactically similar features, rather than features that match on the basis of meaning. A knowledge based matcher is able to use the relations between concepts, their constraints, etc. to explain why two syntactically different features match. In a system for diagnosing starting-problems with cars, for example, the features *no-lights-on-dashboard* and *dead-radio* matches since they both are linked by an *is-caused-by* relation to the *battery-fault* concept.

[1] The field is known as "Case Based Reasoning (CBR)" as well as "Case Based Learning (CBL)". Since reasoning (problem solving) and learning is highly integrated, we propose a merging of the two as the top level term.

The case based approach to reasoning and machine learning (Kolodner-83, Porter-86, Hammond-87, Rissland-87, Stanfill-88) has had a considerable growth during the last couple of years. The first meeting of some size was sponsored by DARPA and took place in Florida in May 1988 (Kolodner-88). Earlier work of importance to this field includes Roger Schank's work on memory structures for learning and reasoning (Schank-82) and the work on transformational and derivational analogy by Jaimee Carbonell (Carbonell-83 and Carbonell-86). Janet Kolodner, a student of Schank's, built one of the first case based learning and reasoning system, called CYRUS (Kolodner-83). More recently, some systems combining the strengths of the case based and the explanation based approaches have emerged (Hammond-88, Bareiss-88a, Barletta-88). In our view this is a particularly promising approach to combining problem solving with knowledge intensive and incremental machine learning. This is also the approach we take in our architecture for expert systems that learn from experience.

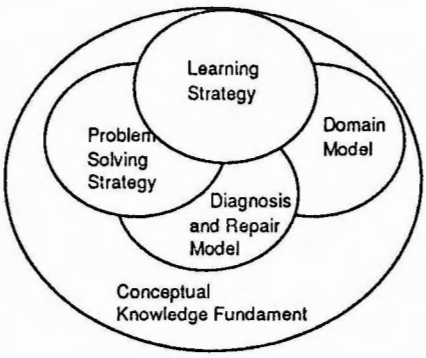
Among the more recent systems, the PROTOS system (Porter-86, Bareiss-88a) - developed at the University of Texas, Austin - is particularly interesting from a KA point of view. It actively pulls the user/expert into the learning process by asking for confirmation of results or supplementary domain knowledge for the case matching and retrieval process. If PROTOS cannot find an explanation for the relevance of a feature to a case, the user is prompted to enter an explanation. In this way general domain knowledge is gradually built up from interaction with the user (end user or expert). This represents a pure bottom up approach to knowledge acquisition. Although domain knowledge may be entered - in the form of object-relationship-object triplets - independent of the cases, this is typically done while entering each case. The representation language is not expressive enough to describe an abstract knowledge- and problem solving model of the domain, integrated with the case structure. As pointed out in (Bareiss-88b), PROTOS' power to generate and heuristically process explanations would greatly benefit from a richer knowledge representation language.

IV. A framework for knowledge based systems

In the following sections an overview of our framework for development of competent, robust and adaptable expert systems are given. We are now developing a system - called CREEK (CbRI through Explicit modelling of Extensive Knowledge) - that instantiates important parts of the framework, and which will be used throughout the chapter to illustrate basic principles and mechanisms. As a test application for developing and refining CREEK, we are developing an expert system for mud diagnosis and treatment. Characteristics of this domain are described in the final section, together with examples of representation of different type of knowledge within our frame-based language.

IV.1 Architecture of the target system

All frameworks for KBS development put some assumptions on the architecture of the target KBS. Our framework supports an architectural structure based on four different operational models: A domain knowledge model, a generic diagnosis model, a problem solving control model, and a learning model.



The main modules

A body of conceptual knowledge forms the basic environment wherein other types of object level and strategic level knowledge is modelled. One unified conceptual model tightly connects the four embedded modules into a coherent knowledge base.

The domain model contains object level knowledge of the domain, while the three ellipses above (see the figure) are control level knowledge used to guide the problem solving and learning processes. The conceptual knowledge fundament glues the models together by being a common reference pool defining all the terms and relations between terms used in the four models. It forms a unified semantic network of all concepts and relations, and can be viewed as a merging of the conceptual models for each of the four inner ellipses in the figure. It may - and indeed should - also contain some knowledge about bordering domains and parts of the general world. The desired level of robustness should always be kept in mind.

The diagnosis&repair (D&R) model defines concept like *evidence*, *observation*, *diagnostic-hypothesis*, *refinement-plan*, etc. It models the problem solving process by a set of actions and their conditioned execution; for example when to apply a particular piece of domain knowledge, when to look for more evidence, when to hypothesise, etc.

The problem solving strategy model controls the use of the D&R and domain models. In particular, it decides what type of knowledge and reasoning method to activate at a given time. The kind of concepts defined at this level are 'D&R-model', 'concept-network', 'rule', 'case', 'model-based-reasoning', 'case-based-reasoning', etc.

The learning model basically contains the algorithms for knowledge-supported case based learning, and a collection of rules to guide the matching and generalisation processes.

IV.2 Knowledge representation

Type of knowledge

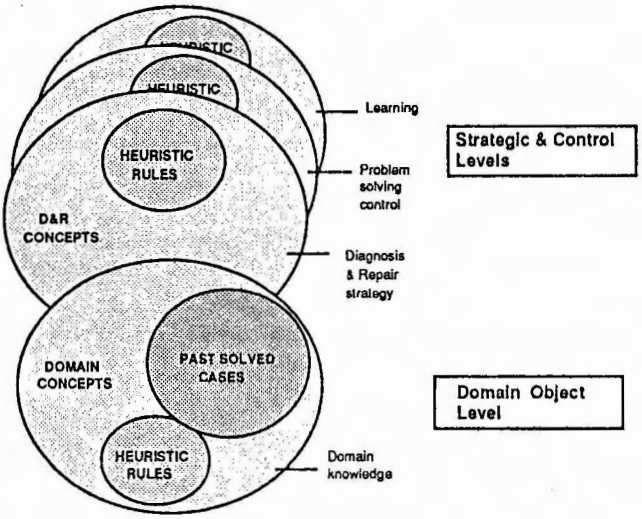
The figure on next page shows what types of knowledge that are represented within the architecture.

The domain object level defines domain concepts and describes facts and assumptions about the domain. In coherence with this conceptual domain knowledge model, heuristic rules may be defined, and past cases will continuously be added by the learning module. Since the learning is case-based, the main part of specific problem solving knowledge is represented as cases. The role of rules is partly to act as support knowledge for the explanation process, partly to be the operational problem solving knowledge to fall back on when no matching case is found for a new problem. Knowledge about how to use the domain knowledge for problem solving and learning is described at the strategic and control level. The definitions of concept and relations at this level is represented as partial conceptual models - all being parts of the unified conceptual knowledge model.

In order to capture the diversity of knowledge types present in our framework, we need a flexible representation formalism. The semantics of the representation language itself (knowledge structure) has to be clearly defined, to be able to express the meaning of the actual knowledge that is represented (knowledge content). We have found a frame system formalism to be the representation that best fulfils our requirements.

The knowledge represented in the system constitutes a unified network of tightly connected frames. Every symbol (entity, attribute, relation, value) is a concept, defined in a separate frame. The meaning of each concept is defined through the concept's relationships with other concepts. This bottoms out on a set of primitives defined by Lisp functions. There are four basic types of frames: standard frames (describing general object concepts), slot frames (describing relation concepts), rule frames and case frames. The two latter have their own reasoning methods

associated with them.



Knowledge representation at the different levels

The figure shows the four levels at which knowledge is described within the framework. The conceptual model at each level is a partial model of the one single fundamental knowledge model. Upon this conceptual knowledge core, heuristic rules is used to represent the more operational parts of the strategies and control knowledge, while past cases constitutes the primary source of operational knowledge at the domain object level.

Our initial approach is to use cases only for domain level (object level) knowledge. The explicit knowledge at the control & strategic levels are represented as conceptual models supported by heuristics. The idea of strategy-cases for the internal control is interesting, though, but not pursued at this stage.

Representation language

In CREEK all types of knowledge is represented within a network frames. The frame language has frame-types for entity concepts, relational concepts, cases and rules. The knowledge content of a system, i.e the basic understanding that the system has of its domain, is embedded in the network of concept definitions. The frame representation system - called FRICK - has been developed as an extension of the frame system used in METAKREK (Sølvberg-88) - a methodology and toolkit for interactive knowledge acquisition. A frame contains slots with facets and values. Slots describe the properties of a concept, i.e. its attributes and its relations with other concepts (frames). The frames form a structural hierarchy implied by the set of structural relations that connect the frames. Four types of structural relations are defined: subclass-of, instance-of, part-of and member-of. They define a parent-child relation between general classes (e.g. car subclass-of vehicle), an individual exemplar of a class (car54 instance-of car), a component description of an object (wheel part-of car) and set membership (my-car member-of norwegian-cars), respectively. A facet may be a value-type (typical-value, default-value), value-constraint (value-class, number-of- entries, value-format), action-type (if-needed, if-updated) or user-defined.

A set of inference methods on the frame system define the basic interpretation of the knowledge structure. So far, a few relatively simple mechanisms for spreading-activation, inheritance, default reasoning and forcing of constraints have been implemented. There are build-in inheritance methods for structural relations like *subclass-of* and *instance-of*. A more flexible and

powerful inheritance method (plausible reasoning method) is under development, where the user may define inheritance related to any kind of transitive relation (e.g. ows-money-to, works-for, likes-the-face-of). A set of *context dependent* heuristics for plausible reasoning (Cohen-88) will also be incorporated into this inheritance mechanism, e.g:

If concept-1 *causes* concept-2	If concept-1 *part-of* concept-2
And concept-1 *part-of* concept-3	And concept-1 *enables* concept-3
Then concept-3 *causes* concept-2	Then concept-2 *has-ability* concept-3

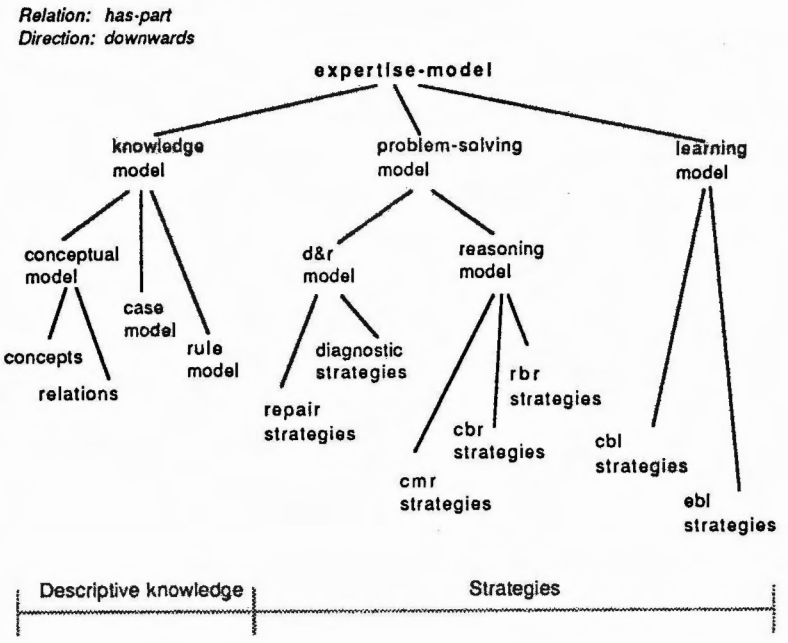
An example, expressed in FRICK simple-access forms (<framename>.<slotname>.<value>) may look like:

```
(bacteria.causes.infection AND bacteria.part-of.dirty-water
 -> dirty-water.causes.infection)
(wheel.part-of.car AND wheel.enables.movement
 -> car.has-ability.movement)
```

FRICK has a graphical frame editor (a modification of METAKREK's concept editor) enabling visualisation and easy definition of frames and frame structures on a TI Explorer. The editor generates CommonLisp code.

IV.3 A model of expertise

As the figure below shows, we view *expertise* as a combination of the abilities to solve problems and to learn how to do it. A fundament for expertise behaviour is an extensive knowledge model within which problem solving as well as learning takes place.



Top level expertise model

The figure shows our view of expertise as a an integration of problem solving and learning based on several types of knowledge, and describes some relevant concepts in a part-of hierarchy.

The knowledge model contains factual and assumptional knowledge of concepts and their attributes, relations and constraints, 'compiled knowledge' in the form of if-then heuristics, and a collection of past experienced problem solving cases.

In the following we will focus on two central components of the framework: The *reasoning model* and the *learning model* (see the figure).

IV.4 Reasoning control.

A main function of the top level problem solving control model is to decide what type of knowledge and reasoning to use for problem solving, given an input case and a goal.

When presented with a new problem, the controller receives a problem frame containing the input features and the current goal. In a system for diagnosis of car troubles, for example, the initial goal could be Find-starting-fault. If a feature is known to the system, it will have a frame that describes its current properties (e.g. parent class, constraints, relations, associated faults, cases reminded of). If an unknown feature is presented, the system asks for a description of the feature, and creates the necessary frames. The system checks whether this description violates any constraints, assumptions or current facts. The features are forward propagated through the domain knowledge model, in a spreading-activation manner, whereby associated concepts (frames) are activated. The context for guiding this process is determined by the current goal and state of the diagnosis&repair model. The goal is continuously updated by the instantiated diagnosis model as the problem solving process proceeds.

This process starts with extraction of relevant features from the input case, and halts when either a criterion is produced for deciding whether to solve the problem by retrieving a former case or activate the rule base, or when a plausible solution to the initial goal is reached. If the features (input or derived) give a reminding to a previous case that is above a particular strength level, case based problem solving is tried, otherwise the rule base is activated. The strength threshold level depends on the relative contents and power of the case based and rule based subparts. The initial value is set explicitly, and adjusted according to how well the chosen problem solver performs. If either the case base or the rule base fail to produce a result, the controller re-evaluates its previous decision given the current state of the system, and updates the threshold value.

IV.5 Case based reasoning and learning.

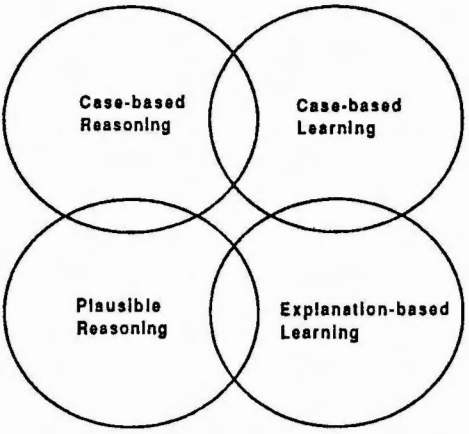
CREEK derives several of its basic ideas from the PROTOS system developed by Bruce Porter and Ray Bareiss (Bareiss88b). The differences are significant, however: PROTOS is a pure exemplar based problem solver, where classification becomes a process of finding the best match to an existing exemplar (stored case). CREEK integrates case based problem solving with problem solving based on more general knowledge, it has a more expressive knowledge representation language allowing a richer body of knowledge to support the reasoning and learning methods, it allows for explicit generalisation of cases, and it has a mechanism to forget old cases of presumably little value.

Cases are stored in a particular type of frame, which inherits slots from the control level concept called "case". Cases form a hierarchical structure of case-classes and actual, individual cases. Case-classes are collections of cases with common relevant features. Individual cases may represent a single past case or a generalisation of several single cases. Cases may be generalised if they have a non-discriminating feature in common with a similar new case. Depending on the importance of the feature it may be deleted or generalised by climbing one of the structural links. All features stored with a case has to be explained, i.e. justified by the general knowledge in the system or by the user/expert. Features are categorised as necessary, characteristic, non-characteristic and irrelevant. The first three are relevant features that are stored if the explanation path linking them to a diagnostic fault is strong enough.

The case based problem solver is activated when CREEK gets a reminding to a previous case, and the strength of the reminding is above some threshold level. A reminding is a 'chunked' explanation path - a kind of macro operator connecting a feature (observed or derived) with a particular case or case-class. The strength of a reminding is calculated by combining the contribution from each of the single explanation steps in the path. The strength of a single explanation is determined by the type of relation connecting the feature to the fault. Independent

causal relations have the highest strength, a co-occurring relation the lowest. Relations may be qualified by "always", "often" and "sometimes" (for example: Flat-battery always causes Starter-motor-does-not-turn), leading to different strengths. Remindings from several features are combined in searching for a matching case. Previous cases may set up expectations that have to be checked for the new case. If an expected value cannot be verified by the system - either directly from the input description or through the frame that defines the feature concept - the user is asked for confirmation or supply of additional information.

The system learns by retaining cases, either directly or merged into a generalised case. If a retrieved case leads to a successful conclusion, this is recorded in a separate slot leading to a strengthening of the reminding from each of the relevant features to the case. A failed case retrieval leads to a weakening of the reminding from the features to the fault of the retrieved case. As the system gains experience, more new cases will be merged with existing cases, leading to a flattening of the growth curve for the knowledge base. To ensure an eventually non-increasing knowledge base, we are working on a forgetting mechanism that will throw out some cases that have not been retrieved over some period of time.



Knowledge intensive case based reasoning and learning

The figure shows the interplay between concrete problem solving cases and more general knowledge in reasoning and learning. The terms 'Plausible reasoning' and 'Explanation based learning' are used as pointers to the kind of research areas that are relevant to our approach: Generation and use of plausible explanations to support case based problem solving and learning.

An interesting extension to the case based learning method would be to learn heuristic rules from the more deep knowledge represented in the conceptual knowledge model (Steels-86). Operational problem solving heuristics could be 'chunked' by 'compiling' relevant explanation chains (chains of plausible arguments) guided and focused by the problem solving goal.

V. Application

Our test application is an expert system for diagnosis and treatment of mud during off-shore drilling[2] . While drilling for oil and gas, the drilling fluid (mud) serves several purposes: Cleaning

[2] In cooperation with Bjørnung Jensen, SINTEF, and Pål Skalle, University of Trondheim.

of the bore hole by removing drilled cuttings, etc., controlling the pressure in the well, lubricating and cooling the drill string and bit, maintaining mechanical stability of the bore hole, bringing constituents of the rock formation to the surface for analysis. Mud engineering is a separate discipline of well drilling and are most often handled by special mud companies. A mud engineer is continuously supervising the mud properties during drilling, adjusting the properties mechanically (e.g. filtering) and chemically (adding chemicals) if necessary. Maintaining the right pressure is extremely important since blow out always is a danger when drilling into a new formation stratum.

V.1 Nature of the problem

Diagnosis and subsequent treatment of mud is a complicated classification problem, from an AI point of view. Diagnosis and treatment are not two separate problems, the latter succeeding the former. The process is rather a merging of partial diagnosis and treatment activities. Typically, some initial observations lead to formation of a diagnostic hypothesis (best guess) for which a treatment is initialised. During treatment more observations and measurements are gathered, possibly leading to a refusal (or specialisation) of the current hypothesis and to the formation of a new one. Since the only way to gather more information about the problem is to start treatment (add chemical additives) based on an initial hypothesis, this treatment must be carefully done in order not to create dangerous situations if the hypothesis should fail. This means that unless the diagnostic hypothesis is pretty certain, other possible hypotheses must also be taken into consideration when a treatment is applied.

The mud itself is a complex liquid containing roughly 20 important constituents. They give the mud its properties described by approx. 40 observable or measurable descriptions. Some of the descriptions are simple property-value pairs (e.g. density 1.4), while others are complex structures (e.g. filter cake description). There are about 20 or so significant single causes for non-normal mud properties (different types of contaminants, formation fluid influx, etc.), and a lot of relevant combinations. A mud engineer supervising the process typically monitors 10-12 properties continually or periodically during normal operation. The values of mud properties are (of course) highly interdependent. Their normal ranges also depends on variables like the depth of drilling, and properties of the formation being drilled through - including local geology at the drilling site.

A knowledge model describing the relations between normal and abnormal values of single mud properties, groups of mud properties, the surrounding formation, and causes and consequences of failure situations, is very difficult to acquire. Nevertheless, some kind of deep model of the domain will be necessary in order to cope with unexpected situations. Because of the complexity of the problem, it is not possible to build a complete domain model of the problem. In order to continuously adapt and improve the system, manual (interactive) knowledge refinement methods, as well as techniques that enable the system itself to automatically learn from its own problem solving activities is needed.

V.2 Examples of knowledge types

Object level

* A conceptual/relational domain model describes what drilling mud is, its functionality, components, mud properties and effects of mud additives. All relevant inter-relationships between concepts - like hierarchical structures, functional relations, causalities, constraints, contextual dependencies - are explicitly described. For example:

FRAME rock-salt		
subclass-of	value	contaminant, entity
part-of	value	rock-formation
causally-increases	value-class	mud-property
	value	chloride-content
causally-decreases	value-class	mud-property
	value	pH
possibly-increases	value-class	mud-property
	value	funnel-viscosity, yield-point, gel-strength
never-changes	value	mud-volume

FRAME salt-water-influx

subclass-of	value	liquid-flow, process
effective-component	value	salt-water
failure-in-case	value	0068, 0121, 0019
consequent-in-rule	value	salt-water-influx-rule

FRAME causally-increases

instance-of	value	modifying-relation-type, causal-relation-type
makes-sense-for	value	mud-component
value-entry	value	measurable-entity
result-type	value	yes/no
evaluation-depends-on	value	val*1 val*2
evaluation-expression	value	(cond ((and (> val*2 val*1) (> (time-stamp val*2) (time-stamp val*1))))

* A collection of past problem solving cases. This part of the KB will be used for case based problem solving: Given an input problem, a past case similar to the present problem is retrieved and used to solve the problem. After a problem has been solved, it may be stored as a new case in the case base. So this part of the knowledge base will be continuously updated during problem solving by a method that retains cases that are not well matched by existing cases.

FRAME case0068

instance-of	value	solved-case
funnel-viscosity	value	(62 high)
	value-dimension sec.	
solids-content	value	(15 a-little-high)
	value-dimension percent	
chloride-content	value	(25000 very-high)
	value-dimension (mg per l)	
failure	value	salt-water-influx
contaminant	value	salt-water
failed-additives	value	fresh-water
successful-additives	value	caustic-soda, lignosulphonate
activated-for	value	case0089, case 0008, case 0152
successful-concl-for	value	case 0008
failed-conclusion-for	value	case 0152, case 0089

* A collection of heuristic rules expressing rules-of-thumb kind of experiential knowledge, i.e. surface level knowledge not captured by the deeper domain model. It may also be knowledge that exists at a deeper level, but is compiled ('cached') to a surface level in order to improve performance.

FRAME saltwater-influx-rule

instance-of	value	domain-rule, rule-of-thumb
if-then	value	((if (ph.state-of-change.value.decreased)) (then (diagnostic-hypothesis.moderate-belief.value.salt-water-influx)))

Control level

* A problem solving model of mud diagnosis and treatment guides the process from the indication of an abnormal situation through fault finding and treatment. This model defines control level concepts (observation, measurement, diagnosis, hypothesis, evidence, cause, etc.), and a hierarchy of actions (collect-more-data, apply-non-risky-additive, etc.) controlled by an overall strategy for mud diagnosis&treatment (when to do what, what to look for, etc.).

FRAME strategy-rule

subclass-of	value	rule-of-thumb
has-instance	value	stepwise-interpretation-rule, check-expectations-rule

```
FRAME stepwise-interpretation-rule
  instance-of          value    strategy-rule
  if-then              value    ((if (possible-abnormality and not all-data-available)
                                 (then (stepwise-interpretation))))

FRAME stepwise-interpretation
  subclass-of          value    strategy
  action-sequence      value    (1 (check funnel-viscosity))
                                (2 (check yield-point))
                                ...
```

VI. Current state of affairs

As previously mentioned, the frame representation system with basic inference methods is implemented (TI Explorer, CommonLisp), and parts of a previosly developed mud model (briefly described in Sølvberg-88) is being tranformed and entered.An extension of METATOOL's Concept Editor (Aakvik-88) is incorporated to assist the conceptual knowledge modelling work. A core part of the initial implementation will be methods that make the conceptual object knowledge model, the control levels and the case base cooperate efficiently. Particular emphasis is on techniques to generate and use supporting explanations in this process.

VI. Evaluation

Evaluation of knowledge acquisition and machine learning techniques has to be done with respect to the goals of the system being developed. Criteria like "quality of knowledge" and "speed of learning" are fundamental, but hard to define. Since the purpose of this framework is to enable building and maintenance of real world expert systems, evaluation will be based on four basic criteria:

* Expressive power, i.e. is the expert able to model all knowledge relevant for the application? Does the results of reasoning methods lead to inferred knowledge which capture the intended meaning?

* Expert interaction. Does the framework give the expert sufficient flexibility and power in adding (integrating) new knowledge and refining the existing knowledge? Are the knowledge models sufficiently transparent?

* Learning method. Is the case based learning method powerful enough? Is the knowledge based matching process able to make sufficient use of the more general knowledge (concept model, rules)? A minimal criterion is that a mistake once made should not be repeated.

* Performance of target system. Does the system solve the relevant problems at the necessary level of competence? Is the knowledge sufficiently robust? A pilot system needs to be tested against the performance of human experts.

VII. Conclusion

In order to develop the next generation of real world expert systems, two important system properties must be focused: A strong and robust competence, and an ability to continuously learn from problem solving experience.The approach to interactive knowledge acquisition, and the techniques for machine learning and problem solving integrated in our framework have separately shown promising results. By combining them in the way shown here, there are strong reasons to believe that the methods will mutually strengthen each other, thereby enabling development and maintenance of more robust and adaptive knowledge based systems for real world environments.

Acknowledgements

Several of these ideas emerged during a visiting year in the University of Texas at Austin. Special thanks to Bruce Porter for many interesting discussions and useful pointers relevant for this work, to Ray Bareiss for discussions on case based reasoning in general and Protos in particular, and to Art Souther, James Lester, Ken Murray and Karl Branting for many enlightening discussions on the role of a strong domain knowledge model in supporting case based and explanation based reasoning and learning.

REFERENCES

Aakvik-88
Geir Aakvik, Mette Vestli, Inge Nordbø, Ingeborg Sølvberg, Tore Amble, Agnar Aamodt: METAKREK; METATOOL documentation. ELAB-RUNIT Report STF14 A88055. SINTEF, Trondheim. 1988.

Bareiss-88a
Ray Bareiss, Bruce Porter, Creg Wier: PROTOS - an exemplar based learning apprentice. 2nd Knowledge acquisition for knowledge based systems workshop. Banff, Canada, 1987. Also in: Int. J. Man-Machine Studies, Vol. 29, 1988, pp. 549-561.

Bareiss-88b
Ray Bareiss: PROTOS; a unified approach to concept representation, classification and learning. Ph.D Dissertation, University of Texas at Austin, Dep. of Comp. Sci. 1988. Technical Report AI88-83.

Branting-88
Karl Branting: The role of explanations in reasoning from legal precedents. In: Kolodner-88, pp. 94-103.

Breuker-87
Joust Breuker, et. al.: Model driven knowledge acquisition; interpretation models. Deliverable (report) Task A1. Esprit project 1098. University of Amsterdam, 1987.

Carbonell-83
Jaime Carbonell: Learning by analogy - formulating and generalizing plans from past experience. Machine Learning, Voll. s.137-161. 1983. Morgan Kaufmann.

Carbonell-86
Jaime Carbonell: Derivational analogy. Machine learning, Vol2. s.371-392. 1986. Morgan Kaufmann.

Clancey-84
William Clancey: The epistemology of a rule based expert system - a framework for explanation. Artificial Intelligence, Vol.20, 1983. pp.215-251.

Cohen-88
Paul Cohen, Cynthia Loiselle: Beyond ISA: Structures for plausible inference in semantic networks. AAAI-88, pp. 415-420.

DeJong-86
Gerald DeJong, Ray Mooney: Explanation based learning - an alternative view. Machine Learning (journal), Vol.1 s.145-176. 1986.

Gruber-88
Thomas Gruber: A method for acquiring strategic knowledge. In: Proceedings of the 3rd Knolwedge Acquisition for Knowledge-based Systems Workshop, Banff, Canada. November 1988. 21 pgs.

Hammond-87
Kristian Hammond: Explaining and repairing plans that fail. IJCAI-87. pp. 109-114.

Hammond-88
Kristian Hammond, Neil Hurwitz: Extracting diagnostic features from explanations. In (Kolodner-88), pp. 169-177.

Keravnou-86
E.T. Keravnou, Les Johnson: Competent expert systems. Kogan Page Ltd. 1986.

Kolodner-83a
Janet Kolodner: Maintaining organization in a dynamic long-term memory. Cognitive Science, Vol.7, s.243-280. 1983.

Kolodner-83b
Janet Kolodner: Reconstructive memory, a computer model. Cognitive Science, Vol.7, s.281-328. 1983.

Kolodner-87
Janet Kolodner: Extending problem solver capabilities through case-based inference. Proc. 4th workshop on Machine Learning, UC-Irvine, June 22-25 1987. pp 167-178.

Kolodner-88
Janet Kolodner (ed.): Case based reasoning. Proceedings from a workshop, Clearwater Beach, Florida, May 1988. Morgan Kaufmann Publ.
Lenat-86
Doug Lenat, M.Prakash, M.Shepherd: CYC - using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks. AI Magazine, Winter 1986, pp. 65-85.
Lenat-87
Doug Lenat, Edward Feigenbaum: On the thresholds of knowledge. IJCAI 1987. s.1173-1182.
McDermott-86
John McDermott: Making expert systems explicit. In: Proceedings of the 10th IFIP Congress, Dublin,1986.
Mitchell-82
Tom Mitchell: Generalization as search. Journal of Artificial Intelligence, Vol 18, no.2, March 1988. pp. 203-226
Mitchell-86
Tom Mitchell, R. Keller, S. Kedar-Cabelli: Explanation based generalisation - a unifying view. Machine Learning (journal), Vol.1 s.47-80. 1986.
Murray-88
Kenneth Murray: KI-1; an experiment in automatic knowledge integration. University of Texas at Austin, Dep. of Comp. Sci. Technical Report AI88-90.
Porter-86
Bruce Porter: PROTOS; an experiment in knowledge acquisition for heuristic classification tasks. Proceedings of the First Intern. Meeting on Advances in Learning, Les Arcs, France. July 1986. pp. 159-174. Also: UT, AI-Lab, Tech.report AI-TR-85-03. 1986.
Quinland-86
J. Ross Quinlan: Induction of decision trees. Machine Learning (Journal), Vol.1, s.81-106. 1986
Rissland-87
Edwina Rissland, Kevin Ashley: HYPO, a case based reasoning system. University of Massachustes, Amhearst. Dep. of Computer and Information Science. The Counseler Project, Project Memo 18. 1987.
Schank-82
Roger Schank: Dynamic memory. Cambrigde University Press. 1982.
Schank86a
Roger Schank, Gregg Collins, Lawrence Hunter: Transcending inductive category formation in learning. Behavorial and Brain Sciences, Vol.9, 1986. pp. 639-650. (pages 651-686 contain commentaries by several authors).
Schank-86b
Roger Schank: Explanation patterns - understanding mechanically and creatively. Lawrence Erlbaum, 1986.
Shepherd-88
Mary Shepherd: Tools for adding knowledge to the CYC LSKB. MCC Technical Report No. ACA-AI-068-88. 1988.
Smith-81
Edward Smith, Douglas Medin: Categories and concepts. Harvard University Press. 1981.
Stanfill-88
Craig Stanfill, David Waltz: The memory based reasoning paradigm. In Kolodner-88. pp.414-424.
Steels-86
Luc Steels. Walter van der Welde: Learning in second generation expert systems. In J.S.Kowalik (ed): Knowledge based problem solving. Prentice Hall 1986. pp 270-295.
Steels-87
Luc Steels: The deepening of expert systems. AICOM-The European Journal on Artificial Intelligence, Vol.0, no.1, August 1987. pp. 9-16.
Sticklen-87
J. Sticklen, J. Smith, B. Chandrasekaran, J.Josephson: Modularity of domain knowledge. International. Journal of Expert Systems, Vol.1. no.1, 1987. pp. 1-15.
Sølvberg-88
Ingeborg Sølvberg, Inge Nordbø, Mette Vestli, Tore Amble, Geir Aakvik, Jorun Eggen, Agnar Aamodt: METAKREK - Methodology and tool-kit for knowledge acquisition. ELAB-RUNIT Report STF14 A88046. SINTEF, Trondheim. 1988.
Wielinga-86
Bob Wielinga, Joust Breuker: Models of Expertise. Proceedings from ECAI-86, Brighton, England. 1986. pp. 306-318.
Wittgestein-53
Ludvig Wittgenstein: Philosophical investigations. Blackwell, 1953. pp. 31-34.

A KNOWLEDGE-BASED SYSTEM THAT SUPPORTS EXPLANATION-BASED LEARNING APPLICATIONS

Lawrence F. Chachere*
Department of Computer Science
University of Illinois
405 N. Mathews
Urbana, IL 61801

Abstract

A major focus in the area of expert system learning is the improvement of knowledge base refinement techniques. ProHC, a reimplementation of HERACLES, aims to improve the automation of knowledge base refinement and to support a variety of complex refinement methods. The key towards these goals lies in the representation of the diagnostic strategy. The reimplementation emphasizes an explicit, modular, declarative knowledge representation that allows the knowledge base to be used interchangeably among learning and teaching subsystems, in addition to the general problem solving inference engine.

1. Background

ProHC (**Prolog Hueristic Classification**) is an expert system shell that has evolved from the MYCIN family of expert systems. MYCIN [Shortliffe, 1976] is one of the earliest successful knowledge-based systems. It acts as a medical consultant for patients with bacterial infections. It uses production rules as its knowledge representation scheme, and features a backward chaining inference engine that is separate from its knowledge base. With the development of GUIDON [Clancey, 1982], a tutoring program that runs on MYCIN, it became apparent that MYCIN was insufficient for teaching and learning on the knowledge base. MYCIN's knowledge representation method complicates these processes because it is difficult to see how the knowledge and production rules interrelate.

The shortcomings of MYCIN led to the development of NEOMYCIN [Clancey and Letsinger, 1981]. The major change in NEOMYCIN is the dichotimization of the knowledge base into problem solving control knowledge and domain knowledge. MYCIN rules were reconfigured so that the domain knowledge is separated out and represented more declaratively. The control knowledge formed a heuristic expert system shell called HERACLES [Clancey, 1986]. This new version of MYCIN better captures the problem-solving steps of an expert, alleviating the problem of teaching on the knowledge base. However, the development of ODYSSEUS [Wilkins, 1985] uncovered limitations in the learning capabilities of NEOMYCIN.

ODYSSEUS is an apprenticeship learning program which works in conjunction with the HERACLES shell. It can implement learning by watching in order to improve a HERACLES-based expert system. The learning program tries to isolate gaps in the knowledge base by observing consultations and trying to construct explanations for questions asked by the human expert. Whenever the system fails to construct an explanation, it is assumed that a learning opportunity is present. The system will suggest a repair and try to validate it.

Despite significant success of experiments with ODYSSEUS, the structure of HERACLES imposes limitations to learning systems. The data structures in HERACLES are inflexible. The shell is not well-designed to handle procedures required by inference engines, other than the procedures needed by the general problem solver for performing consultations. Furthermore, partially due to the fact that NEOMYCIN is a product of several years of development by many different researchers, the system is very hard to understand. The code has never been completely documented and is difficult to build upon.

* Current affiliation: AT&T Bell Laboratories, Naperville, IL. 60566

The rest of this paper discusses ProHC, a reimplementation and refinement of HERACLES. Section 2 describes the architecture and knowledge representation of the new system, and how it differs from HERACLES. Section 3 discusses how this representation contributes toward the goal of implementing explanation-based learning subsystems. Section 4 describes two types of explanation-based learning that have been applied to ProHC. The conclusion lists several areas for future work with ProHC.

2. An Overview of ProHC

2.1 Architecture of ProHC

The development of ProHC has centered on two architectural goals. The first is the multiple use of control knowledge. Given a particular class of problem (e.g., diagnostic, synthesis, etc.), a common body of strategic knowledge is applicable to any given domain. The second goal is the multiple use of domain knowledge. Humans have the ability to make use of knowledge about a domain in many different manners, regardless of whether the activity involves solving a problem, teaching, or learning from accumulated knowledge.

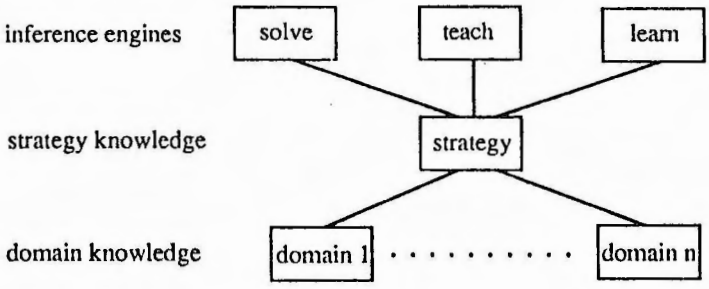


Figure 1: Subsystem Hierarchy

Figure 1 depicts a knowledge-based system as a hierarchy of subsystems. ProHC is a *strategy* subsystem, containing knowledge for solving diagnostic problems. This strategy knowledge has also been referred to as *meta-level knowledge* [Davis 1976]. The medical knowledge from NEOMYCIN has been translated into a domain knowledge subsystem [Tan, 1989] that can used in conjunction with ProHC. *ProHC-Med* is the knowledge base consisting of the new shell and the translated medical knowledge.

The hierarchy in Figure 1 has three levels: an inference level, a strategy level, and a domain level. The domain and strategy levels collectively comprise the knowledge base. An expert system having a knowledge base that is separable from its inference engine is known as a knowledge-based system. MYCIN falls into this category. The major difference between the architectures of NEOMYCIN and MYCIN is that MYCIN does not have separable strategy and domain levels. One of NEOMYCIN's advantages of having distinct strategy and domain subsystems is that its control knowledge is not restricted for use on the medical domain. CASTER [Thompson and Clancey, 1986], a program that diagnoses defects in cast iron, was built upon the HERACLES shell.

NEOMYCIN, however, does not totally fit the architectural description of Figure 1. Its interface between the strategy knowledge and inference engines is not completely interchangeable. The subsytems in the inference level represent three classes of problems: 1. the use of information within the knowledge base (solving), 2. the transfer of information from the knowledge base to an external source (teaching), and 3. the transfer of information into the knowledge base (learning). While HERACLES is adequate at addressing the first two types of problems, the third is the main objective of ProHC.

2.2 Knowledge Representation in ProHC

Figure 2 is a diagram of the ProHC-Med knowldge-base. As previously mentioned, it consists of two subsystems. The domain subsystem contains two forms of knowledge, facts and rules. Domain facts are relations that contain simple frame knowledge. An example is *suggests(nausea,migraine)*, meaning 'nausea suggests a migraine.' Domain rules are represented as Horn clauses and contain numerical associations between etiologies and findings. *rule(rule2740, conclude(meningitis,.8),[finding(seizures,yes),finding(tense_fontanel,yes)])* is a domain rule that means 'conclude with a degree of certainty of .8 that the patient has meningitis if the patient has seizures and the patient's anterior fontanel is abnormally tense or bulging.'

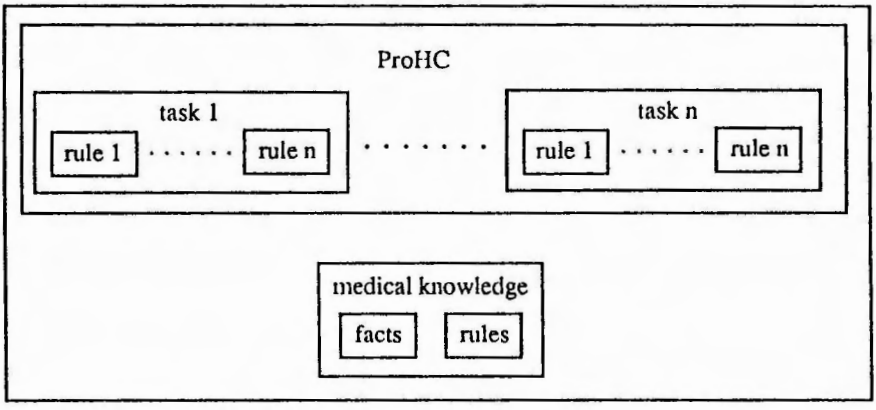


Figure 2: The ProHC-Med Knowledge Base

The strategy knowledge in ProHC consists of a hierarchy of tasks. Each task is accomplished by an ordered set of rules. The rules in this subsystem are referred to as *metarules* [Davis 1976]. A metarule has the following format:

goal :- precondition$_1$, ..., precondition$_n$, action(s).

A precondition is a relation that accesses either static information from the domain subsystem or dynamic, problem-state information that is generated during a consultation. An action is either a call to a subtask, an assertion or retraction from the global database of problem state information, application of a domain rule (which also affects the global database), or a user-interface operation.

The representation of metarules is the most significant difference between HERACLES and ProHC. The metarules in ProHC are Horn clauses encoded in Prolog. Those in HERACLES are frame structures encoded in MRS, a derivative of LISP. The following HERACLES metarule and its ProHC translation serve to illustrate representational differences between the shells. The example chosen is the second metarule from the task *clarify_finding*. This rule causes the system to findout about process questions for a finding that is known to be present. For example, if *headache* is being clarified, the user will be prompted for *headache_duration* and *headache_severity*.

The HERACLES rule is:

```
(PUTPROPS RULE088
      PREMISE  (MAKESET (CLARIFY.QUESTIONS FOCUSDATUM) FOCUSQ)
      ACTION  (FORLOOP FOCUSQ (TASK FINDOUT $PROCPARM) $PROCPARM)
      TASK  CLARIFY-FINDING
```

```
SUBJECT TASKRULES
TRANS ((VAR ASKINGPARM)
        "is a characterizing feature of" a (VAR FOCUSDATUM))
RULEHISTORYVARS ($PROCAPARM)
MRSRELATIONS [((CLARIFYQ $FOCUSDATUM $PROCPARM)
                (AND (PROCESSQ $FOCUSDATUM $PROCPARM)
                     (NOT (TRACEDP ROOTNODE '$PROCPARM]
MRSPREMISE (MAKESET (CLARIFY.QUESTIONS FOCUSDATUM
                        $PROCPARM) $FOCUSQS)
INPUT-VARS ((SFOCUSDATUM FOCUSDATUM
                     (CLARIFYQ $FOCUSDATUM $PROCPARM)))
OUTPUT-VARS ((SPROCPARM FOCUSQS
                     (CLARIFYQ $FOCUSDATUM $PROCPARM)))
DOMAIN-RELS ((PROCESSQ $FOCUSDATUM $PROCPARM))
DYNAMIC-RELS [(NOT (TRACEDP ROOTNODE 'PROCPARM])
```

The same rule in ProHC is:

```
goal(clarify_finding(Finding1),cf2) :-
    not value(Finding1,no),
    not value(Finding1,unknown),
    clarified_by(Finding1,Finding2),
    not concluded(Finding2),
    goal(findout(Finding2)).
```

The ProHC version is much more concise and considerably less difficult to understand. The explicitness of the metarules in ProHC can be attributed to three policies:

1. Strong emphasis is placed on descriptive variable and predicate names. For example, 'Finding1' and 'Finding2' are considerably more descriptive about the pieces of knowledge they represent than their HERACLES counterparts, '$FOCUSDATUM' and '$PROCPARM'. The difference can be even more conspicuous when comparing entire clauses, e.g., 'not concluded(Finding2)' vs. 'NOT(TRACEDP ROOTNODE $PROCPARM)'.

2. Very few premises have attachments to complex procedures. Most premises simply check a domain fact or state variable. The only exceptions are premises that involve universal quantification, e.g., *max_differential(Hypothesis)*, unifies with the hypothesis on the differential having the highest certainty factor. It is computed by comparing the strengths of all hypotheses that unify with the differential predicate. There is no way to expand such routines within Horn clauses, without the use of extra-logical constructs.

3. Control structures are not present in the metarules. Because so little explicit control is available in Prolog, it has an advantage over LISP as a declarative language. The non-use of extra-logical constructs, such as cut and fail, forces a concentration on the explicit representation of the knowledge required to solve the problem.

There are two other important restrictions placed on the ProHC metarule set. Premises are never allowed to trigger side-effects that change the problem state. Furthermore, a general requirement concerning metarules with multiple actions is that all actions are triggered uniformly. The results of executing an action are never a factor in determining whether another action in the same rule should be triggered. The significance of this sharp distinction between premises and actions is elaborated upon in the next section.

3. Qualities of an Inference-Independent Shell

The ability for a ProHC-based expert system to support a variety of different inference engines lies chiefly in the representation of the ProHC shell. Complex inference engines, in particular the explanation-based learning subsystems that will be described in the next section, place strict

demands on a knowledge base. Three characteristics about the metarule data structures in ProHC facilitate the development of complex inference engines.

A *declarative* metarule representation is most beneficial for teaching purposes. However, the declarativeness of the strategy level is also important to all other types of applications; if the shell is easy to understand, it will also be easy to build upon. The effort towards making the metarules declarative has been discussed in the previous section. The success of this effort is evident by the fact that students have successfully been able to produce semester class projects using the system. In contrast to ProHC-Med, it takes much longer than a semester to understand NEOMYCIN.

A learning by watching program, described in the next section, can be implemented if the metarule representation is *reversible.* An explanation-based learning by watching subsystem involves a backward inference engine that can run the metarules in reverse. The learning by watching scenario involves the construction of explanations for problem-solving actions of an expert. Given the sequence of questions asked by the expert, along with the responses, the learning engine will try to derive the steps that led to each question.

The other explanation-based learning subsystem that will be described in this paper, a learning by doing program, relies on an *introspective* metarule representation. The scenario for this type of learning involves analysis of cases that the system misdiagnoses. In order to explain a misdiagnosis, the learning engine monitors metarule failures during the course of problem solving. Because premise failures determine the cause of a metarule failure, it is important that the system can extract premise queries without triggering side-effects. For this reason, premises and actions within the metarules are clearly separated.

A final point worth mentioning is the contribution of the Prolog language toward the inference-independence of the shell. The data-program equivalence in Prolog makes it very easy to design inference engines through the use of meta-programming. The inference engine for solving a problem using the ProHC-Med knowledge base is an extension of a basic meta-interpreter described in *The Art of Prolog* [Sterling and Shapiro, 1986].

4. Knowledge Base Refinement on ProHC-Med

This section describes several learning subsystems that have been implemented using ProHC-Med. These subsystems are all directed towards refinement of domain knowledge. Under the assumption that the knowledge in the shell is accurate and complete, any pathology within the system can be attributed to the domain level.

Two general knowledge base refinement techniques have been applied to ProHC-Med. A case-based refinement approach uses simple heuristics to modify the knowledge base. The modifications are based on comparisons between a case library of successful diagnoses and diagnoses made by the expert system on the same set of cases. Case-based refinement methods are discussed in the first subsection.

The second refinement approach is explanation-based. The use of explanation-based learning to improve a knowledge base is not new. LEAP [Mitchell et al., 1985] is a learning system that has the capability of explaining training instances. Learning is accomplished by generalizing these explanations. However, the type of explanation-based learning that is implemented on ProHC is different. A learning opportunity occurs when the domain theory of the system is incapable of producing an explanation of a training instance [Wilkins 1989]. The failure indicates that the domain knowledge is incomplete, and learning occurs by filling in the knowledge gap.

Two explanation-based subsystems are described in this section. Section 4.2 discusses explanation-based learning by watching. Section 4.3 discusses explanation-based learning by doing.

4.1 Case-Based Refinement

Systems using the case-based refinement approach [Tan, 1989] can be directed toward domain theory pathologies: invalid knowledge, and missing knowledge. In all cases, *global credit assignment*, the determination that there is a problem with the domain knowledge, occurs when the solving engine misdiagnoses a case. Upon discovering that a pathology exists, the system will try to isolate the problem (*local credit assignment*) and suggest a repair. Repairs are verified if they result in a higher number of correct diagnoses.

One refinement method aimed at an invalid domain theory works under the assumption that an erroneous domain rule has an inaccurate certainty factor (CF). In ProHC-Med, CFs range from -1 to 1. A positive value supports belief in the conclusion, whereas a negative value supports disbelief in the conclusion. A CF of zero indicates that the premise is unrelated to the conclusion.

The initial CFs of the domain rules in NEOMYCIN were obtained by asking the domain experts to supply them. This is not a very reliable method because domain experts generally do not solve problems using precise numerical weights, and will thus have difficulty in deriving such values. A more formal way of deriving numerical values is through Bayesian induction. By using a case library that includes finding values for each case, in addition to the set of final diagnoses, empirical values can be computed for each domain rule. Bayes' equation for computing the probability of a particular hypothesis given a particular finding is: $P(H/F) = [\ P(F/H)\ P(H)\ ]\ /\ P(F)$, where H is the hypothesis and F is the finding.

There are a variety of ways that the empirical CFs can be figured into the set of domain rules. They can either be weighted in with the current CFs or, because the original CFs were not mathematically derived, the existing CFs could simply be discarded. Experiments on ProHC-Med used the latter method.

A refinement method that addresses missing knowledge problem is based on the distribution of the domain rules among the set of diseases. Local credit assignment can be determined by making comparisons between the number of occurences of a given disease and its relative frequency among the conclusions of the domain rules. It is likely that there are missing domain rules about a given disease if that disease is often diagnosed *false negative*, i.e., the disease is diagnosed negative when it is actually present. An extreme case in ProHC-Med is an hypothesis in the system that is not supported by any of the domain rules. Whenever *primary_brain_tumor* is the correct diagnosis of a library case, ProHC is unable to conclde this disease due to the absence of rules in the domain knowledge that can support it. A repair can be made by inducing new rules based on the examples in the case library that diagnose the disease.

The previous method does not work when the missing knowledge takes the form of overspecialized or overgeneralized domain rules. There may be plenty of rules about a given hypothesis in the system, but few of them may be very useful. An overspecialized rule is one which is seldom triggered due to a large number of premises within the antecedent. If such a rule has a very high CF, subrules can be created by dropping one or more of the conditions from the antecedent and using Bayes' formula to generate CFs. The new subrules are added to the system if they are not overgeneralized. An overgeneralized rule is one that is triggered often, but has such a low CF that it does not make a significant contribution toward diagnosis. These rules can be specialized into stronger rules by adding conditions to the antecedent. These conditions are induced from examples in the case library.

Case-base methods have several weaknesses. One is inductive bias, e.g., what should be the threshold that determines whether a rule is overgeneralized or overspecialized? But more importantly, the reliance on a case library can tempt learning to focus on statistics rather than the causes of the problems. Refining a domain theory to fit a particular case library will not necessarily improve its performance on other case libraries. Indeed, while case-based refinement experiments on ProHC-Med gave it the ability to diagnose correctly some previously misdiagnosed cases, it also lost its ability to properly diagnose other cases.

Nevertheless, case-based methods have proven very effective in improving ProHC-Med. Experiments were performed using a library of 112 solved cases that were obtained from records at

the Stanford Medical Hospital. While ProHC-Med was initially able to correctly diagnose 32 of these cases, 68 correct diagnoses resulted from a cumulative application of the three types of case-based refinement that have been discussed [Wlikins and Tan, 1989]. The improved accuracy from 28.5% to 60.7% gives an aggregate improvement of 32%.

Although the refinement systems discussed in this subsection are an effective means of improving a ProHC-based expert system, they do not reflect the intent of the restructuring of the shell. The heuristics for local credit assignment and repair are not affected by the structure of control knowledge. Most expert systems have the ability to handle global credit assignment for these methods, because all that is required is a simple forward engine for problem solving. In contrast, the explanation-based methods in the following subsections are either very difficult or impossible to implement on most knowledge bases.

4.2 Learning by Watching

A subsystem for learning by watching on ProHC-Med is modelled after an algorithm for apprenticeship learning in ODYSSEUS. Apprenticeship learning is a form of learning by watching that occurs as a byproduct of building explanations of human problem solving. An apprenticeship is the most powerful method that human experts use to refine and debug their expertise in knowledge-intensive domains such as medicine [Wilkins, 1989].

The processes of knowledge base deficiency detection and repair rely on an inference engine that runs the control knowledge metarules backwards. This engine is assisted by the generation of *backward goals*. For each metarule in the shell, Horn clauses are created for each action in the right-hand-side of the rule:

$$\text{bgoal(action,goal)} :- \text{precondition}_1, \ldots, \text{precondition}_n.$$

The clarify-finding metarule example from section 2 generates one backward goal:

```
bgoal(findout(Finding2),clarify_finding(Finding1),cf2) :-
        not value(Finding1,no),
        not value(Finding1,unknown),
        clarified_by(Finding1,Finding2),
        not concluded(Finding2).
```

Whenever the system observes an expert asking a question, it tries to find explanations for this action by chaining through the backward goals. A question unifies with an 'ask_user' action within the task *findout*. If the system cannot instantiate a path to the top-level task (*consult*), an explanation failure occurs and it is assumed that the knowledge base is missing a piece of information that the expert knows.

In order to conjecture a repair, the system makes the single fault assumption. Candidate repairs are derived by backward chaining through the backward goals, in the attempt to construct a proof that will explain why a question was asked. Conjuncts in the right-hand-side of a backward goal represent either problem state knowledge or domain knowledge. During the process of backward chaining, if a unification failure occurs because of conflicting problem state knolwedge, the path is rejected. However, the a unification failure is caused by a missing domain knowledge, that piece of domain knowledge can be temporarily added to the database. Under the single fault assumption, a proof is allowed to supply only one missing domain tuple along the entire path.

Candidate repairs are validated by a confirmation theory. The confirmation theory assigns decision procedures to domain knowledge predicates and heuristic rules. The decision procedures determine whether a conjecture is acceptable. The following two Horn clauses form a decision procedure for determining whether a trigger rule instantiation is acceptable. An English translation of the procedure is, "a trigger rule has only soft data within its antecedent."

```
trigger_rule(Rule) :- not unacceptable_trigger_rule(Rule).
```

```
unacceptable_trigger_rule(Rule) :- has_premise(Rule,Finding), not soft_datum(Finding).
```

A good way to illustrate the functionality of the learning by watching engine is by showing how a specific question is processed. Consider when an expert asks, "Does the patient have eyes that are red and painful?" If it has already been asked and confirmed that the patient has a headache, or if the system can infer that the patient has a headache, the following explanation gets generated by the backward engine:

```
==> findout(red_painful_eye)
==> applyrule_backward(rule2740)
==> process_finding(headache)
==> forward_reason(headache)
==> identify_problem
==> make_diagnosis
==> consult
```

Suppose that trigger_rule(Rule2740) were retracted from the knowledge base. Also assume that the system is in a problem state in which the system determines that there is not yet sufficient evidence for 'glaucoma' to be on the differential (this would be the only other possible cause for asking about red_painful_eye). Under these circumstances, the system would have no explanation for the question. While trying to suggest a repair, the following proof would be constructed:

```
bgoal(findout(red_painful_eye),applyrule_backward(Rule),ab2) :-
        not rule_applied(rule2740),
        has_premise(rule2740,red_painful_eye),
        not concluded(red_painful_eye).

bgoal(applyrule_backward(rule2740),process_finding(headache),pf2) :-
        in_premise(headache,rule2740),
        not rule_applied(rule2740),
        trigger_rule(rule2740),
        premise_true(rule2740).

bgoal(process_finding(headache),forward_reason(headache),fr2) :-
        new_datum(headache),
        finding(headache).

bgoal(forward_reason(headache),identify_problem,ip1).
```

The above chain of explanation defines a candidate proof because the trigger_rule relation is the only uninstantiable tuple. The above proof happens to be the only candidate proof that gets generated under the given conditions. Because 'trigger_rule(rule2740)' fulfills the decision procedure, it will be reasserted to the knowledge base.

Experiments on the case library have demonstrated success with this type of learning. The experiments were performed using only one confirmation theory; a procedure that defines acceptable 'process_subtype' tuples. Applying the learning by watching subsystem to the 44 cases that were still misdiagnosed after application of the case-based methods resulted in a gain of 5 additional properly diagnosed cases [Wilkins and Tan, 1989].

4.3 Learning by Doing

The susbsystem for learning by watching on ProHC-Med also focuses on missing knowledge. Unlike the previous system, however, it does not use a backward engine. Instead, it uses an extended version of the forward engine for problem solving.

The type of misdiagnosis that determines global credit assignment for learning by doing is different from the misdiagnosis defintion used by the other learning subsystems. In the other subsystems, misdiagnosis is defined as a difference between a known answer in a case library, and the answer produced by ProHC-Med, the strongest hypothesis on the differential after execution terminates. However, a misdiagnosis in learning by doing pertains more to a failure to come up with a diagnosis, rather than coming up with a wrong diagnosis. An inductive bias is used to classify misdiagnosed cases. The bias defines a properly diagnosed case to be one in which the strongest hypothesis has a degree of certainty higher than some particular threshold.

The assumption for learning by doing is that a failure to diagnose a case results from a failure to perform some particular action. Local credit assignment can therefore be accomplished by monitoring metarule failures in the course of problem solving. However, some inductive bias is necessary in order to focus on particular metarule failures. A single consultation will result in an enormous number of metarule failures, yet the vast majority of these failures are not learning opportunities.

For this reason, one particular metarule, the first metarule in the task *differentiate_hypotheses* (dh1), has been chosen as a focus for learning by doing:

```
goal(differentiate_hypotheses,dh1) :-
        differential(Hypothesis1),
        differential(Hypothesis2),
        different(Hypothesis1,Hypothesis2),
        suggested_by(Hypothesis1,Finding),
        suggested_by(Hypothesis2,Finding),
        not concluded(Finding),
        evidence_for(Hypothesis1,Finding,Rule1),
        evidence_for(Hypothesis2,Finding,Rule2),
        differentiating_rules(Rule1,Rule2),
        goal(findout(Finding)),
        goal(forward_reason(Finding)).
```

The translation of this metarule is, "findout about and forward reason upon, all findings that discriminate between all the possible pairs of different hypotheses on the differential." Two hypotheses can be discriminated only by a unique piece of data that can be considered as evidence for both hypotheses. Thus, there must be two rules within the domain knowledge that use a common finding to make conclusions about them. Two domain rules can differentiate hypotheses if, given a particular finding common to the antecedents of both rules: 1. The finding-value that suggests one hypothesis is different from the value that suggests the other hypothesis, or 2. A particular finding value supports one hypothesis while discrediting the other.

To help illustrate the functionality of dh1, the following scenario is an example of how this rule gets applied to the medical domain knowledge. Suppose the hypotheses 'allergic_sinusitis' and 'cluster_headache' were both on the differential in the current problem state. The bindings that occur upon solving dh1 are:

```
Hypothesis1 --> allergic_sinusitus
Hypothesis2 --> cluster_headache
Finding --> headache_location
Rule1 --> rule1770
Rule2 --> rule1620
```

The unification succeeds because the two rules are triggered by different values of 'headache_location'. A 'frontal' headache suggests allergic sinusitis, whereas a 'unilateral' headache suggests a cluster headache. Now suppose that either 'rule1620' or 'rule1770' were removed from the system. A learning opportunity would then exist because the knowledge base knows of no other finding that differentiates between the two hypotheses.

Local credit assignment for learning by doing proceeds as follows:

1. Select a case that ProHC-Med misdiagnoses.

2. Whenever dh1 is tried, keep track of the premise instantiations.

3. If dh1 fails, determine the cause of failure:
   a. If 'different(Hypothesis1,Hypothesis2)' fails, there is no learning opportunity.
   b. If all premises succeed except for 'not concluded(Finding)', there is also no opportunity for learning.
   c. Otherwise, there is a hypothesis pair for which a discriminating finding does not exist. A learning opportunity is present.

4. When the failure is due to the third reason, record the pair of undifferentiable hypotheses.

There are several approaches that can be taken in order to generate a repair. The approach used by the subsystem implemented for ProHC makes use of a case library. The repair will consist of a new domain rule. Candidate repairs are generated by the following procedure:

1. Select one of the hypotheses.

2. Determine all the findings that suggest the selected hypothesis.

3. For each suggestive finding, induce a set of rules:
   a. For every possible finding value, construct a rule where: the premise asserts that the finding has the given value, and the conclusion is the unselected hypothesis.
   b. Determine CFs by applying Bayes' formula to the case library.
   c. Select the rule with the highest CF.

Verification of a repair is accomplished by temporarily adding it to the knowledge base, and rediagnosing the case. A repair is considered valid if the diagnosis concludes an hypothesis that has a degree of certainty exceeding the threshold. No experimental data using the case library, however, is available yet.

5. Conclusion

The explanation-based learning subsystems that ProHC-Med supports have the advantage that they automate a reasoning process for determining local credit assignment. It is apparent that the explanation-based methods tend to generate fewer solutions, e.g., compared to case-based methods. However, because they focus on the true causes of the pathologies in a knowledge base, there is a greater probability that the repairs suggested by explanation-based subsystems will be accurate.

There are quite a few areas that remain to be explored. One unaddressed area is explanation-based subsystems that correct false information within a knowledge base. The two subsystems in this paper are only capable of learning missing information. Another problem is to automate refinement on control knowledge. One such project, PRONEO64 [Najem, 1989], demonstrated great difficulties with refining this type of knowledge using the current ProHC shell. Finally, work can be done to analyze, improve, and verify inductive biases for explanation-based learning methods.

Acknowledgements

I would like to thank Prof. David C. Wilkins, my M.S. thesis advisor, for contriving this open-ended research topic that will lead to many future discussions. I would also like to thank the members of the knowledge-based systems group who have helped in either developing or analyzing the system: Charlene Bloch, Ziad Najem, Young-Tak Park, and Kok-Wah Tan. Finally, I would like to express my appreciation to Sheldon Nicholl for assisting my efforts to work on the system and obtain data and references while I have been off-campus.

References

Clancey, W. J. (1982). GUIDON. In Barr, A. and Feigenbaum, E. A., editors, *The Handbook of Artificial Intelligence, Vol. 2*, pages 267-278, Reading, MA.: Addison-Wesley.

Clancey, W. J. and Bock, C. (1986). Representing Control Knowledge as Abstract Tasks and Metarules. In Coombs, M. and Bolc, L., editors, *Computer Expert Systems*, Springer Verlag.

Clancey, W. J. and Letsinger, R. (1981). NEOMYCIN: reconfiguring a rule-based expert system for application to teaching. In Clancey, W. J. and Shortliffe, E. H., editors, *Readings in Medical Artificial Intelligence: The First Decade*, chapter 15, pages 361-381, Reading, MA.: Addison-Wesley.

Davis R. (1976). *Applicstions of meta-level knowledge to the construction, maintenance, and use of large knowledge bases.* HPP Memo 76-7 and AI Memo 283, Stanford University.

Mitchell, T. M., Mahadevan, S., and Steinberg, L. I. (1985). LEAP: a learning apprentice for VLSI design. In *Proceedings of the 1985 IJCAI*, pages 573-580, Los Angeles, CA.

Najem, Z. (Najem). *The Representation and learning of control knowledge for a generic Expert System shell.* Master's thesis, University of Illinois at Urbana-Champaign.

Shortliffe, E. H. (1976). *Computer Based Medical Consultations: MYCIN.* Elsevier.

Sterling, L. and Shapiro, E. (1986). *The Art of Prolog: Advanced Programming Techniques*, chapter 13. The MIT press.

Tan, K. (1989). *Representation and Refinement of a Heuristic Classification System.* Master's thesis, University of Illinois at Urbana-Champaign.

Thompson, T. and Clancey, W. (1986). A qualitative modelling shell for process diagnosis. *IEEE Software.*, 3(2):6-15.

Wilkins, D. C., Clancey, W. J., Buchanon, B. G. (1985). An Overview of the Odysseus Learning Apprentice.

Wilkins, D. C. and Tan., K. (1989). Knowledge Based Refinement as Improving an Incorrect, Inconsistent and Incomplete Domain Theory. To be presented at the Machine Learning Workshop, Ithaca, NY.

On Explanation and Bias in Inductive Concept-Learning.

*Luc De Raedt, Maurice Bruynooghe**

Department of Computer Science
Katholieke Universiteit Leuven
Celestijnenlaan 200A
B-3030 Heverlee
Belgium

ABSTRACT

Most concept-learning approaches require that the user completely specifies each example, which means that each possibly relevant feature of examples must be explicited in the description language of the examples. We propose an alternative approach which involves indirect relevance. In our approach, the user only has to supply the directly involved objects in an example and the system will automatically compute the relevant relations between these objects. Together these relations form an explanation of the example and determine the description language of concepts. As the initial explanation may be incomplete, it is possible to extend it by shifting the bias [Utgoff 86]. In contrast to existing approaches, our technique allows to shift the bias of first-order languages. We also show how our method can be integrated in concept-learning systems. We integrated the method in the interactive concept-learner CLINT [De Raedt 88ab]. The resulting system requires only a minimum amount of information from the user. Hence, it is very promising for use in Learning Apprentice Systems.

1. Introduction.

Most concept-learning approaches (e.g. [Mitchell 82],[Quinlan 86],[Kodratoff 88] ,...) require that the user completely specifies each example, which means that for each example one has to provide all possibly relevant features explicitly to the system. This property of concept-learning problems is called *direct relevance*. A concept-learning problem is *indirect relevant* [Michalski 83] if it is not *direct relevant*. Direct relevance is a strong requirement for concept-learning systems as it charges the user with a non-trivial and tedious task : he must not only identify all possibly relevant features for a concept-learning problem but also describe all examples in terms of these features. This task can clearly be difficult, expensive and time-consuming. Direct relevance also implies that an important aspect of the concept-learning problem, i.e. the choice of the description language, has already been solved (cf. [Utgoff 82],[Mitchell 80], [Utgoff 86],[Rendell 86]).

Despite the importance of indirect relevance and its potentially powerful applications, the problem of indirect relevance has not received much attention in the machine learning literature. A few notable exceptions are : DISCIPLE [Kodratoff 87],[Tecuci 88] where the system proposes features to the user which must be classified by the user as relevant or irrelevant for the concept-learning problem; AQ11 [Michalski 83] and MARVIN [Sammut 86] where the user has to supply the system with a set of basic features from which additional features may be deduced using background knowledge and PGA [Buntine 87] where it is possible to cope with indirect relevance within a predefined language. Finally, there is the interactive concept-learner CLINT [De Raedt 88a], which also copes with indirect relevance within a predefined language. Here, we extend CLINT in order to handle indirect relevance completely by providing mechanisms to shift the bias.

The problem of indirect relevance is also related to the problem of justification (of the bias) and explanation (of examples). Explanation is a topic of special interest for artificial intelligence (consider the research on EBL [Mitchell 86],[DeJong 86], the explanation facilities in expert systems [Hayes-Roth 83], the definition of artificial intelligence as the science of explanations [Kodratoff 87]). In terms of justification and explanation, the problem of indirect relevance may

* Supported by the National Fund for Scientific Research (Belgium).

be specified as follows : given a positive (respectively negative) example of an unknown concept and domain-knowledge, find a set of features of the example which explain why it is a positive (resp. negative) example for the concept, and which determine the bias of the concept.
We propose a solution for the problem of indirect relevance, in which the user only needs to supply the objects which are directly involved in an example and the system derives a set of relevant relations between these objects specifying an explanation for the example. During the explanation process, the user can (but need not) help the system, by telling the system which relations in a proposed explanation are irrelevant for the concept. This information results in computational advantages, since it reduces the search-space for the concept. As the initial explanations may not be sufficient for the concept, we also discuss *how* and *when* it should be extended. Our solution is applicable for conjunctive as well as disjunctive concepts. Extending an explanation has a similar effect as a shift of bias [Utgoff 82], [Utgoff 86] of the concept-description language. Under certain conditions it is guaranteed that a correct explanation will be found. The resulting method can directly be applied in general concept-learning systems like DISCIPLE [Kodratoff 87],[Tecuci 88], MARVIN [Sammut 86], CLINT [De Raedt 88a], versionspaces [Mitchell 82], ...
As an application of our technique, we show how it is integrated in CLINT [De Raedt 88a]. The resulting system [De Raedt 88b] is very promising as it minimizes the information which the user has to supply to learn a concept. As CLINT is interactive, it generates almost all the examples it needs. In this way all examples supply new information to the system; the number of examples is minimized and the user is freed of the burden of collecting the examples. Furthermore, the examples only specify the objects which are directly involved. So, the user need not consider the relations holding between these objects. In this way the user's task is reduced to classifying the examples as positive or negative. Because of these two features, CLINT is very promising for use in Learning Apprentice Systems which operate in a weak, incomplete or imperfect domain-theory like e.g. DISCIPLE [Kodratoff 87], [Tecuci 88].
In section 2 we give a specification of the problem and in section 3 we present a solution which is valid for conjunctive as well as disjunctive concepts. The solution is formulated in a logic framework [Genesereth 87] and shifts the bias when necessary [Utgoff 86]. The integration of the technique of indirect relevance in CLINT is discussed in section 4. In section 5, we formulate some conclusions and touch briefly upon related work.

2. A problem-specification.

With the term *concept-learning* we refer to the process of generating a concept-description from examples. In this paper, our definitions are formulated in a logical framework [Genesereth 87],[Sterling 86]. Within the logical framework a *concept* is a *predicate*, a *concept-description* is a set of *Horn-clauses* for the predicate and *examples* are ground instances of a predicate. A *clause* is a set of *literals*. A *literal* is a positive or negative atom in first order logic. We will restrict ourselves to literals in which no functional expressions occur. A *Horn-clause* is a clause in which at most one positive literal occurs. We denote the positive literal of a clause c by head(c) and the set of positive literals corresponding to the negative literals by body(c). A notation for a Horn-clause[1] c used in Prolog is head(c):-body(c). The semantics of a Horn-clause are such that the head is true whenever all literals in the body are true. In this way the body forms a sufficient set of conditions for the head. A Horn-clause *covers* a ground instance g if there is a unifier θ such that head(c) unifies with g and all literals in body(c)θ are true. For more details we refer to [Genesereth 87], [Lloyd 87], [Sterling86].
In practice the concept-learning process is not only driven by training-examples but also by bias [Utgoff 82], [Utgoff 86], [Mitchell 80], which is -following Utgoff- anything which influences how

1) We assume that literals are of the form p(X1, ... , Xn) with the Xi distinct variables. This is not an additional restriction because constants are allowed in the knowledge-base and an equality predicate is also available e.g. q(a,X,X) rewrites to q(X,Y,Z) :- A(X), Y = Z where A(a) is in the knowledge-base.

the concept-learner draws inductive inferences based on the training-examples. The significance of bias for concept-learning is widely recognized [Utgoff 82], [Utgoff 86], [Mitchell 80], [Ganascia 88]. Although bias may be incorporated in a concept-learning system in a variety of ways (cf. [Utgoff 82]), we will only consider bias in the form of restrictions on the description language.

In our framework, a bias is *correct* for learning a predicate p if there exists a set of Horn-clauses S for the predicate p in the description language associated with the bias, such that all positive and no negative examples are covered by S. We also distinguish between disjunctive and conjunctive concepts. If a concept can be described in one Horn-clause it is a *conjunctive* predicate. Otherwise it is *disjunctive*. Knowledge about the disjunctive or conjunctive nature of a concept is also a form of bias which will be considered here.

Our notion of concept-learning is defined precisely in figure 1.

Given
- a set of positive examples (i.e. ground instances) of an unknown predicate p
- a set of negative examples (i.e. ground instances) of the predicate p
- a description language in which to define the predicate p
- a knowledge base of predicate definitions (other than p) which may be used in the definition of p and which specify information about the objects in the examples

Find
- a definition for p which is in the description language and which
- is complete (i.e. covers all positive examples) and consistent (i.e. covers no negative examples) (cf. [Mitchell 82])

Figure 1 : a definition of concept-learning.

This definition of concept-learning is unusual in the sense that the examples only specify the directly involved objects and not the relations between these objects. Note also that in contrast to EBL [Mitchell 86],[DeJong 86], we are doing induction because there is no definition of the predicate p given in our initial knowledge-base ! As a matter of fact, it is the aim of the learning process to induce such a definition.

An example illustrating the above definition is given below.

Example 1.

The knowledge base.

```
place(p1).      composition(p1,concrete).   color(p1,white).
place(p2).      composition(p2,sand).       color(p2,blue).

table(t1).      composition(t1,metal).      color(t1,red).      weight(t1,50).
table(t2).      composition(t2,wood).       color(t2,grey).     weight(t2,50).

block(b1).      composition(b1,wood).       color(b1,red).      weight(b1,20).
block(b2).      composition(b2,plastic).    color(b2,blue).     weight(b2,40).
block(b3).      composition(b3,metal).      color(b3,red).      weight(b3,30).
block(b4).      composition(b4,stone).      color(b4,grey).     weight(b4,120).
block(b5).      composition(b5,glass).      color(b5,white).    weight(b5,40).

X = X .

less(X,Y) :- X < Y .

object(X) :- place(X).
```

object(X) :- table(X).
object(X) :- block(X).

The positive examples.

same_color(t2,b4).
same_color(b2,p2).

The negative examples.

same_color(t2,b5).
same_color(b1,t2).

The description language 1.

Our first description language is the language of *completely bound Horn-clauses.* A Horn-clause c is *completely bound* if there are no constants in c and all variables which occur in body(c) also occur in head(c). The concept same_color cannot be described in the language of completely bound Horn-clauses. Indeed, the most specific clause covering the first positive example is same_color(X,Y) :- table(X), block(Y), object(X), object(Y) and it also covers the first negative example.

The description language 2.

If the description language of concepts is full Prolog, the concept same_color can be described as follows : same_color(X,Y) :- color(X,XC), color(Y,YC), XC=YC.
In a data-driven concept-learner, there are two steps in processing an example : in a first step the *explanation*[2] is computed and in a second step the versionspace of concepts is adjusted. An explanation of an example is a most specific element in the language associated with the bias, such that the explanation covers the example. An explanation of same_color(t2,b4) in the language of completely bound Horn-clauses is shown above.
From the example one can see that an inappropriate bias may prevent the concept-learner from finding a complete and consistent concept-description. However, bias is important and necessary. Without bias one cannot decide which properties to include in an explanation, because all properties in the knowledge base are considered equally important. In our example less(6,7) would be considered equally important as color(t2,grey). One might want to solve the problem by including all the properties in the explanation, but this is not feasible because then one should include all relations in the knowledge base. So, bias is necessary to restrict the number of relations to include in an explanation. However, because of the restrictions, it may prevent the learner from identifying the concept. Therefore, it should be possible to change, i.e. to shift, the bias when such a situation occurs. As a result, the ideal bias is a flexible one like in [Utgoff 86] which allows both to identify the most relevant explanations and to relax its restrictions whenever necessary.

3. A solution for the concept-learning problem.

In this section we propose a technique to solve the problem of indirect relevance. First we show how a concept-learner can learn a concept given a certain bias or language. Then we present some interesting languages which can be used to describe concepts. In fact, different series of languages are presented. The elements of the series are ordered according to growing expressiveness. The use of the language-series is shown and it is discussed how and when the bias should be shifted. Finally we indicate how the user can assist the system in order to optimize the search. During the discussion we distinguish between conjunctive and disjunctive concept-learning problems. In section 4 we show how these techniques are integrated in the system CLINT [De Raedt & Bruynooghe 88ab].

---

2) Throughout this paper, we use the term *explanation* for a set of conditions which *may* indicate why an example belongs to a concept. In contrast to EBL [Mitchell 86],[DeJong 86], our notion of explanation does not mean that the conditions are sufficient for the concept. Perhaps, the term *plausible explanation* would be more precise. However, for brevity we use *explanation*.

3.1. How to learn a concept given a language-schema ?

We assume that a language is completely defined by a language-schema L, and that language-schema's can be used by the system. Some language-schema's are shown in the next section. A language-schema L and a knowledge-base determine the versionspace of a concept. We define the functions VS(L,P,N) and DVS(L,P,N) in order to capture this notion. VS is defined for conjunctive concepts and DVS for disjunctive ones. The elements of VS are clauses and the elements of DVS are sets of clauses. The functions take three arguments : a language-schema (determining the bias) and a set of positive and negative examples. Its result is the set of descriptions in the language which cover all positive examples of P and no negative ones of N.

$VS(L,P,N) = \{ c \in L \mid c$ is complete and consistent w.r.t. P and N$\}$
For VS the following relations hold :

$VS(L,\phi,\phi) = L$

$VS(L,P \cup \{p\},N) = VS(L,P,N) \cap VS(L,\{p\},N)$

$VS(L,P,N \cup \{n\}) = VS(L,P,N) - VS(L,\{n\},\phi)$
A special case of the function VS is :

$VS(L,\{p\},\phi)$
.It contains the set of all elements in the
language L which cover p.
This set is completely determined by :

$.EX(L,p) = \{ c \in VS(L,\{p\},\phi) \mid c$ is most specific in $VS(L,\{p\},\phi) \}$.

A clause c is *more specific* than a clause c' iff head(c) = head(c') and body(c') $\subset$ body(c). EX completely determines $VS(L,\{p\},\phi)$ since VS contains all clauses for which there is a clause in EX whose head is the same and whose body is a superset of the body of the clause in VS. The elements of EX are the explanations of the example p. The reason why we call them explanations is that they contain the relations which are valid in the example and indicate which properties are relevant for (i.e. explain why the example belongs to) the concept (cf. also section 3.4).
For disjunctive concepts, the situation is more complex. The function DVS defines the space of complete and consistent *disjunctive* concept-descriptions as follows :

$DVS(L,P,N) = \{ S \mid S$ is a minimal set of clauses in L such that S is complete and consistent for P and N $\}$

With the term *minimal* we mean that there is no real subset of S which is complete and consistent. If a clause is deleted from a minimal concept-description then there will be positive examples which are not longer covered. For DVS the following relations hold :

$DVS(L,\phi,\phi) = \{\phi\}$

$DVS(L,P \cup \{p\},N) = \{ S \cup \{c\} \mid S \in DVS(L,P,N), S \cap VS(L,\{p\},\phi) = \phi , c \in VS(L,\{p\},N)$

and $S \cup \{c\}$ is minimal $\}$

$\cup \{ S \in DVS(L,P,N) \mid S \cap VS(L,\{p\},\phi) \neq \phi \}$

$DVS(L,P,N \cup \{n\}) = \{ S \mid S \in DVS(L,P,N)$ and $S \cap VS(L,\{n\},\phi) = \phi\}$

The rules for VS and DVS are valid for all concept-learners that learn minimal complete and consistent concept-descriptions. They are given in order to make abstraction of specific algorithms and representations. They will allow us to state operational conditions under which to shift the bias and these conditions will be independent of any implementation or system.

3.2. The language-schema's.

For Horn-clause logic we list some useful series of bias below. Each bias corresponds to a language. The list serves only as an illustration and is not meant to be exhaustive. We suppose Horn-clauses which do not contain any constants. (Note, that this is not really a restriction, since defining a predicate for each constant, which succeeds only on that constant realizes the same effect as allowing constants in the clauses.)

Series 1.

$L_i = \{ c = p(X_1, \ldots, X_n) :- body(c) \mid body(c) \subset L_i[(X_1, \ldots, X_n)] \text{ and } body(c) \neq \phi \}$

$L_i[(X_1, \ldots, X_n)] = \{ q(Y_1, \ldots, Y_k) \mid$
q is a predicate-name and
$\#(\{ Y_1, \ldots, Y_k\} - \{X_1, \ldots, X_n\}) \leq i$ and
$\#(\{X_1, \ldots, X_n\} \cap \{ Y_1, \ldots, Y_k\}) > 0 \} \cup \{X_j = X_k \mid j < k\}$

Notice that all Y_1, which are different from the X_j are unique ! In words, the definition means that the conditions of clauses in L_i are allowed to contain at most i variables which do not occur in the head. i is called the variable-degree of a clause.
E.g. with the knowledge-base of example 1 :

$L_0[(X,Y)]$ = { place(X), place(Y), composition(X,Y), composition(Y,X), color(X,Y), color(Y,X), table(X), table(Y), weight(X,Y), weight(Y,X), object(X), object(Y), less(X,Y), less(Y,X), X = Y }

$L_1[(X,Y)]=L_0[(X,Y)] \cup$
{composition(X,Z1), composition(Y,Z2), composition(Z3,X), composition(Z4,Y), color(X,Z5), color(Y,Z6), color(Z7,X), color(Z8,Y), weight(X,Z9), weight(Y,Z10), weight(Z11,X), weight(Z12,Y), less(X,Z13), less(Y,Z14), less(Z15,X), less(Z16,Y) }

From this language, variants may be derived according to whether the equality relation =, is allowed to range over more or less variables.

Series 2.

$L_i^k = \{ c = p(X_1, \ldots, X_n) :- body(c) \mid body(c) \subset L_i^k[(X_1, \ldots, X_n)] \text{ and } body(c) \neq \phi \}$

$L_i^k[(X_1, \ldots, X_n)] = \{ q(Y_1, \ldots, Y_m) \in L_i[(Z_1, \ldots, Z_l)] \mid$
$(Z_1, \ldots, Z_l)$ are all the variables which occur in $L_i^{k-1}[(X_1, \ldots, X_n)] \}$

where
$L_i^1[(X_1, \ldots, X_n)] = L_i[(X_1, \ldots, X_n)]$

In these languages, k defines the *level of existential quantifiers* and i the variable degree of the clause. If k is increased then existentially quantified variables of a deeper lever are allowed. E.g. with the knowledge-base of example 1 :

$L_1^1[(X,Y)]=L_1[(X,Y)]$

$L_1^2[(X,Y)]= L_1[(X,Y,Z_1, \ldots, Z_{16})]$

This means that the following clause is a member of L_1^2 :

p(X,Y) :- object(X), object(Y), color(X,Z1), color(Y,Z2), color(W11,Z1), color(W21,Z2).

Note also that with these languages, we can derive a lot of different series of biases : e.g.

$L_1^1, L_2^1, L_2^2, \ldots$

$L_1^1, L_1^2, L_1^3, \ldots$

Series 3.

$L_{i1, i2 \ \ldots \ ,in}[(X_1, \ldots, X_t)] = \{q(Y_1, \ldots, Y_k) \in L_{in}[(Z_1, \ldots, Z_m)] \mid (Z_1, \ldots, Z_m)$ are all the variables of the members of $L_{i1, i2 \ \ldots \ ,i\,n-1}[(X_1, \ldots, X_t)]\}$

Obviously the sequence is initialized with L_{i1}. For these languages, one can also derive a lot of series. E.g. the following clause is in $L_{0,1,0}$:

same_color(X,Y) :- table(X) , block(Y) , weight(X,XW) , weight(Y,YW) , composition(X,XC) , composition(Y,YC) , color(X,CX) , color(Y,CY) , object(X) , object(Y), CX=CY, less(XW,YW).

For all these languages the following property holds :

Property :

For a language-schema L (belonging to series 1,2 or 3) an example p and a knowledge-base , $VS(L,\{p\},\phi)$ is finite. It is because of this property that we can detect when a certain bias is incorrect. Note that this situation can occur since the languages do not contain facts, i.e. the language of examples is not a subset of the language of concepts. The property also implies that each language can only express a certain class of concepts. Indeed, for each language, one can formulate a concept-description which is not in the language. For such concept-descriptions one must shift the bias. The language-series split an infinite language (a subset of PROLOG) into an infinite number of finite sublanguages.

3.3. When and how to shift the bias ?

In our framework it is easy to characterize an incorrect bias. A bias L is incorrect for learning a conjunctive predicate with examples P and N, whenever $VS(L,P,N) = \phi$. A bias L is incorrect for learning a disjunctive predicate with examples P and N, whenever $DVS(L,P,N) = \phi$. When the bias is incorrect one can shift it. A high-level algorithm to learn concepts and shift the bias is shown in figure 2. The algorithm assumes a (possibly infinite) series of language-schema's

```
i = 1
While there is no solution do

Select an example
Adjust the versionspace (VS(L_i,P,N) or DVS(L_i,P,N)) with a concept-learning algorithm

if the versionspace (VS(L_i,P,N) or DVS(L_i,P,N)) = φ then i = i + 1
```

Figure 2 : meta-learning in the limit.

$L_1, \ldots, L_n, \ldots$ in which to describe the concept. All languages corresponding to the schema's must be finite. Whenever it finds out that the current language-schema does not allow to describe a concept, it tries the next language-schema in the series. Of course, when trying L_i it is unnecessary to consider elements of $L_i \cap \cup_{j<i} L_j$. This kind of knowledge may be used to prune the search. If any of the biases in the series is correct, we shall eventually arrive at a correct bias and be able to learn a correct concept-description. In this way one achieves a kind of

double layered learning in the limit [Biermann 86],[Angluin 82]. Indeed, if there is a correct bias, the above sketched method will identify it and then a sound learner of the second step will identify the correct concept-description. So, our two step learning procedure will not only identify the correct bias but also the correct concept-description. As such learning the bias may be seen as a kind of *meta-learning*. Of course, if the meta-learning process is to be successful in general, the following condition should be fulfilled (it is a necessary condition, not a sufficient one) :

$\forall i \ \exists j > i : L_j - \cup_{k<j} L_k \neq \phi$

Note that the algorithm makes abstraction of how the system searches VS or DVS. Any *sound* concept-learner may be used. Sound concept-learners consider all the elements of the language and are able to detect VS=ϕ. Examples are the versionspace algorithm [Mitchell 82], CLINT [De Raedt 88ab], MIS [Shapiro 82], learning by enumeration [Angluin 83],[Biermann 86].

In the disjunctive case, the conditions under which to shift the bias can be reformulated. The reason is that a disjunctive concept-description may be partially incorrect. Indeed, a disjunctive concept-description may be incorrect because one of its clauses is incorrect. If the method of algorithm 2 is used as it stands then a lot of work is wasted because the correct parts of the concept-descriptions are thrown away together with the incorrect parts. Therefore it is better to use the reformulation of the condition DVS=ϕ in the algorithm and to shift only the language of the incorrect clauses. In case that $\forall$ i $L_i \subset L_{i+1}$ then the modified algorithm will always yield a valid result. Also the reformulation is easier to verify (i.e. more operational) than the original condition. The reformulated condition is used by CLINT (cf. section 4).

Theorem :

The condition DVS(L,P,N)=ϕ is equivalent to

$\exists p \in P$ VS(L,{p},N) = ϕ and also to $\exists p \in P \ \forall c \in EX(L,p)$: c covers an element of N

The result is stated without proof since it is trivial. The reformulated condition is more operational since the number of descriptions to consider is significantly smaller.

3.4. Explanations and optimizations.

In section 3.1. we defined the set of explanations of an example p w.r.t. a language L EX(L,p). The elements of EX(L,p) are explanations because they indicate why p belongs to the concept.
As an example, consider the explanations of same_color(t2,b4) :

EX(L_0,same_color(t2,b4)) = {same_color(X,Y) :- block(Y), table(X), object(X), object(Y).}

EX($L_{0,1}$,same_color(t2,b4)) = {same_color(X,Y) :- block(Y), table(X), object(X), object(Y), weight(X,XW), weight(Y,YW),color(X,XC), color(Y,YC), composition(X,CX), composition(Y,CY).}

EX($L_{0,1,0}$,same_color(t2,b4)) = { same_color(X,Y) :- block(Y), table(X), object(X), object(Y), weight(X,XW), weight(Y,YW),color(X,XC), color(Y,YC), composition(X,CX), composition(Y,CY),XC=YC, less(XW,YW).}

All conditions in the body of the explanations are true for a certain substitution which includes X=t2 and Y=b4. These conditions, i.e. properties of the example together explain same_color(t2,b4).

The explanations may also be used to optimize the search. The optimization technique is based on a similar technique, used in DISCIPLE [Kodratoff 87],[Tecuci 88]. However it goes further because it also allows to use the structure of the language. When explanations are derived for

an example, they may contain irrelevant relations. Such relations increase the search-space (VS or DVS) for the concept-learner of the second step and hence they slow down and complicate the learning process, e.g. by requiring more examples to converge to a solution. Often, the user can identify some of these unwanted relations and allow the system in this way to optimize the search.
Suppose that we have an explanation (a clause) c for an example and that we know that there is an irrelevant relation s occurring as a condition in c. Then this means that we may discard the relation s from our explanation c. This will reduce the search-space VS significantly. The structure of the language-schema may be used to optimize the search further. E.g. if it is known that in $L_{0,1,0}$ for the example same_color(t2,b4), the relations weight(X,XW) and weight(Y,YW) are irrelevant then one can infer that less(XW,YW) is irrelevant. In CLINT, this information reduces the search-space significantly. Instead of requiring 8 examples in order to converge to a solution, CLINT needs only 5 with this irrelevance information.

Irrelevance information can be obtained by querying the user e.g. by presenting the relations in an explanation to the user and ask to the user to classify them as relevant or irrelevant. In a lot of cases it is not only interesting to present these relations to the user but also the substitutions which are true in the example. So instead of merely presenting color(X,XC) one can present color(t2,grey). Other sources of irrelevance information include background-knowledge in the form of e.g. determinations [Russell 86] or relevance constraints [Buntine 87].

In this respect it is interesting to note that one may also use meta-knowledge about the relationship between different predicates in the body of an explanation (cf. [De Raedt 88a]). E.g. if the concept-learner knows that object(X) is more general than table(X), then it needs only to consider hypotheses in VS which do not contain both object(X) and table(X).

3.5. Dynamically computing the next language-schema.

As a matter of fact, the choice of the language-series to be used by the algorithm is a bit arbitrary and might depend on the application. It is not straight-forward how to choose a language-series independent of the application. Therefore it may be useful to compute the next language (or bias) in the series dynamically rather then supposing that it is statically determined. It is easy to devise an algorithm to realize such a dynamic choice. The next language is determined each time that VS or DVS = ϕ. One can imagine that in such a modified algorithm there are different languages to consider as a next bias. In such a case one can search best-first through this space as follows : keep a stack of candidate-biases; initialize it to the next biases; if the stack contains a member which solves the current problem, stop and return that one; otherwise take the best one from the stack and compute its successor(s).

4. An application of the method with CLINT [De Raedt 88ab].

In this section we will briefly describe how the method is integrated in CLINT. CLINT uses the method described in this paper for its first step, and the generalization method of [De Raedt 88a] in its second step. In [De Raedt88b], we present the integrated system without going into detail about example-generation [De Raedt 88a] or bias and language. Here, we will only discuss CLINT w.r.t. to explanation and bias. Instead of presenting the complete CLINT algorithms we will present a detailed example of CLINT. CLINT is very powerful and possesses significant advantages over classical concept-learning systems. First, with CLINT the user has only to supply some examples to the system as most of the examples are constructed by CLINT. These system-constructed examples must then be classified by the user as positive or negative. In this way all examples supply new information to the system and the user is freed from the burden of collecting relevant examples and presenting them to to the system. Secondly, the system allows for indirect relevance, in the sense discussed in this paper. The system is also incremental and suitable for learning multiple concepts, in a way similar to MARVIN [Sammut 86]. Moreover, concept identification in the limit is guaranteed (cf. [De Raedt 88a]), if one of the biases contains a correct concept-description.

```
CLINT =
  As long as there are examples
    Receive an example e for a predicate from the user
    Add e to the set of examples
    Handle_example(e)

Handle_example(e) =
   If e is positive and uncovered then Handle_uncovered(e)
   If e is negative and covered then Handle_covered(e)

Handle_uncovered(e) =
   bias_index = 0
   Repeat
      bias_index = bias_index + 1
      Repeat
        Find an explanation c for e in the language L_bias_index which
         was not considered yet
      Until all explanations in L_bias_index for e have been considered or
            c does not cover negative examples
    Until c does not cover negative examples
    Invoke the generalization method of [De Raedt 88a]
       to generate new examples and to generalize c
    Add the obtained generalization to the knowledge-base

Handle_covered(e) =
   Analyze the proof tree of e in order to obtain an incorrect clause c
   Discard c from the current concept-description
   For (some of) the old positive examples e'
     Handle_example(e')
```

Figure 3 : A summary of the shift of bias in CLINT.

4.1. What CLINT does.

CLINT starts learning from one positive example e, which is supplied by the user. In a first step, it constructs an explanation for the example, expressing the relations between the objects in e. This explanation is used to generate further examples of the concept. These examples are shown to the user who has to classify them as positive or negative. In case an example is positive CLINT generalizes, otherwise it tries to construct other relevant examples. In this way CLINT converges to a clause which covers some positive examples and no negative ones. That clause is asserted in the knowledge-base. When the user later signals that an example is not handled correctly, CLINT will try to recover from this error. If the example is positive (and not covered), a new clause for the concept will be derived using CLINT's basic algorithm (as described above). On the other hand, if the example is negative (and covered), CLINT will identify the guilty clause(s) by asking intelligent questions. These questions are also examples (about concepts which are used in the proof) and they correspond to nodes in the proof tree of the example. Shapiro's method [Shapiro 82] is used to minimize the number of these questions. The culprit(s) is then retracted from the knowledge base and CLINT will verify the known positive examples which were covered by this clause. If any of these are no longer covered, CLINT will enter its error-recovery process again. In [De Raedt 88a], we proved that this method will identify the concept in the limit even for disjunctive concepts, provided that the concept can be described in the language it uses.

The method presented in section 3 is implemented in CLINT. CLINT learns disjunctive concepts

of the form described by DVS. It uses the optimized DVS-condition to detect when to shift the bias. CLINT considers only one clause at a time. Such a clause is a most specific one and it summarizes some of the already processed examples. The reason for considering only one clause at a time is that this is easier for CLINT's generalization and example-generation process. However, when there is more than one most specific clause (e.g. when there is more than one explanation for an example) CLINT can backtrack in order to consider the other possibilities. Alternatively, CLINT could not backtrack and discard some solutions. For a lot of applications this would not necessarily lead to bad results since for some learning problems, the explanations of an example differ only in minor details. When CLINT shifts its bias, it will not reconsider the clauses belonging to the previous languages, even if these also occur in the new language. A summary of CLINT's algorithm w.r.t. bias management is shown in figure 3.

A session with CLINT is given below. The system uses the knowledge-base of example 1 and the language series L_0 , $L_{0,1}$, $L_{0,1,0}$, ... All examples are generated by CLINT, except for the first positive example of a clause and the examples which prove that the bias or the generalization is incorrect. The latter examples must be supplied by the user. So, CLINT (and also the algorithm of figure 2) adopt the following (lazy) strategy : assume that the concept and the bias are correct, unless it has been shown that the opposite is true. In this way the number of user-supplied examples is very small. The number of questions posed by CLINT to the user, is also small (cf. [De Raedt 88a]).

Comments are in italic, user input in bold.

?-learn.
Positive Example of New Predicate ?
@ same_color(t2,b4).

An explanation for the example is computed using the first language schema L_0 . Notice that the explanations contain only relations which hold for the example. With this language and knowledge-base there is only one explanation. Although the system knows that object is more general than block and table, it keeps these predicates in the body in order to facilitate the generalization procedure. This knowledge is however used by the system during the generalization process and in order to simplify final clauses.

Explanation :
```
same_color(X,Y) :-
    table(X) ,
    block(Y) ,←
    object(X) ,
    object(Y) .
```

The system tries to generalize and after considering one candidate generalization, it constructs an example and it asks the user to classify the example.

Is same_color(t1,t1) positive ? [y/n]
@ y.
After searching 1 clauses
Next clause :
```
same_color(X,Y) :-
    table(X) , ←
    object(X) ,
    object(Y) .
```

As the user classifies it as positive, the system generalizes.

Is same_color(b1,t1) positive ? [y/n]
@ y.
After searching 2 clauses
Next clause :
```
same_color(X,Y) :-
    object(X) ,
    object(Y) .
```

The system is unable to find further generalizations, so the clause is asserted in the knowledge-base.

No more valid generalizations
after searching 2 clauses
Generalized clause :
```
same_color(X,Y) :-
```

```
      object(X) ,
      object(Y) .
?-error.
Example which is incorrectly handled ?
@ same_color(t2,b5).
```

The user signals that an error has occurred. The system locates and retracts the incorrect clause by analyzing the proof for the example. There are two possibilities : object is a learned predicate which may contain errors, or object is known to be correct. In the first case it is necessary to ask the two verification questions, while in the second case this is not necessary.

```
Verify whether object(b5) is true ? [y/n]
@ y.
Verify whether object(t2) is true ? [y/n]
@ y.
Retracting wrong clause :
    same_color(X,Y) :-
      object(X) ,
      object(Y) .
same_color(t2,b4) is positive and uncovered
Explanation :
    same_color(X,Y) :-
      table(X) ,
      block(Y) ,
      object(X) ,
      object(Y) .
same_color(t2,b5) is negative and
covered by explanation
```

As the system discovers that the explanation covers a negative example and there are no other explanations of same_color(t2,b4) in the language, it decides to shift the bias. The new language-schema $L_{0,1}$ includes clauses where each condition can have one variable not occurring in the head. No relations between these existentially quantified variables are allowed. Again there is only one explanation.

```
Shifting bias
Explanation :
    same_color(X,Y) :-
      table(X) ,
      block(Y) ,
      weight(X,XW) ,
      weight(Y,YW) ,
      composition(X,XC) ,
      composition(Y,YC) ,
      color(X,CX) ,
      color(Y,CY) ,
      object(X) ,
      object(Y) .
same_color(t2,b5) is negative and
covered by explanation
```

Similarly, the bias is shifted to $L_{0,1,0}$.. This language also allows for relations between the existentially quantified variables of the body. Again there is only one explanation.

```
Shifting bias
Explanation :
    same_color(X,Y) :-
      table(X) ,←
      block(Y) ,
      weight(X,XW) ,
      weight(Y,YW) ,
      composition(X,XC) ,
      composition(Y,YC) ,
      color(X,CX) ,
      color(Y,CY) ,
      object(X) ,
      object(Y) ,
      CX=CY,
      less(XW,YW).
Is same_color(b1,b3) positive ? [y/n]
@ y.
After searching 2 clauses
Next clause :
    same_color(X,Y) :-
      block(Y) , ←
      weight(X,XW) ,
      weight(Y,YW) ,
      composition(X,XC) ,
      composition(Y,YC) ,
      color(X,CX) ,
      color(Y,CY) ,
      object(X) ,
      object(Y) ,
      CX=CY,
      less(XW,YW).
I know same_color(b1,t1) is positive,
so I may generalize !
After searching 1 clauses
Next clause :
    same_color(X,Y) :-
      weight(X,XW) ,
      weight(Y,YW) ,
      composition(X,XC) ,
      composition(Y,YC) ,
```

```
        color(X,CX) ,
        color(Y,CY) ,
        object(X) ,
        object(Y) ,
        CX=CY,
        less(XW,YW). ←
Is same_color(b1,b2) positive ? [y/n]
@ n.
Is same_color(b1,b1) positive ? [y/n]
@ y.
After searching 8 clauses
Next clause :
    same_color(X,Y) :-
        weight(X,XW) ,
        weight(Y,YW) , ←
        composition(X,XC) ,
        composition(Y,YC) ,
        color(X,CX) ,
        color(Y,CY) ,
        object(X) ,
        object(Y) ,
        CX=CY .
Is same_color(b2,p2) positive ? [y/n]
@ y.
After searching 6 clauses
Next clause :
    same_color(X,Y) :-
        weight(X,XW) , ←
        composition(X,XC) ,
        composition(Y,YC) ,
        color(X,CX) ,
        color(Y,CY) ,
        object(X) ,
        object(Y) ,
        CX=CY .
Is same_color(p1,p1) positive ? [y/n]
@ y.
After searching 6 clauses
Next clause :
    same_color(X,Y) :-
        composition(X,XC) ,
        composition(Y,YC) ,
        color(X,CX) ,
        color(Y,CY) ,
        object(X) ,
        object(Y) ,
        CX=CY .
No more valid generalizations
Generalized and simplified clause :
    same_color(X,Y) :-
        color(X,CX) ,
        color(Y,CY) ,
        CX=CY .
```

The final clause can be simplified by the knowledge CLINT gathers during generalization.
This session with CLINT, implemented in BIMprolog took about 50 seconds of CPU-time on a SUN 3/50.

5. Conclusions and related work.

In this paper we presented a general and original approach to cope with indirect relevance in inductive concept-learning. Our technique requires only that the directly involved objects in the examples are specified. The explanation, i.e. the relevant relations between these objects which justify it as a (positive or negative) example, is then automatically derived. Because the description language of concepts may depend on the particular learning task, a domain-independent technique must be able to recover from an incorrect bias. If explanations are computed from the bias by the system, they may not be sufficient because the bias does not allow to describe the concept. Therefore, we incorporated also a technique to shift the bias. An operational criterion to detect when the bias is incorrect and a method to shift the bias were presented. In this context, a shift of bias can be seen as a form of meta-learning. The resulting method is directly suited for use in general concept-learning systems. To support this claim, we showed how it can be integrated in CLINT [De Raedt 88ab], an interactive concept-learner. The combination of both methods leads to an interesting system as it requires only little information from the user, because of the use of indirect relevance and because CLINT constructs most examples itself. Therefore CLINT is very promising for use in Learning Apprentice Systems operating in a weak, incomplete or imperfect domain-theory [Rajomoney 87],[Kodratoff 87], [Tecuci 88].

The presented work is related to a lot of learning techniques like Explanation Based Learning (EBL) [Mitchell 86], [De Jong 86], Learning Apprentice Systems (LAS) [Kodratoff 87], experimentation [Subramian 86],[Sammut 86], shifting the bias [Mitchell 80],[Utgoff 82],[Utgoff 86],[Ganascia 88]; classical concept-learning systems like AQ11 [Michalski 83],MARVIN [Sammut 86], CHARADE [Ganascia 88]; and indirect relevance [Buntine 87]. We will now have

a closer look at these relations.
As our technique finds an explanation for an example, it has a lot in common with EBL [Mitchell 86],[De Jong 86]. The difference is that EBL requires a definition for the predicate in the example, whereas such a definition is not required for our technique. As a matter of fact, computing a definition for the predicate is the aim of the induction process. Moreover, EBL is a deductive process, while we perform induction. The LAS DISCIPLE [Kodratoff 87], [Tecuci 88] has a lot of interesting features. Some of the ideas in CLINT were inspired on DISCIPLE. DISCIPLE [Kodratoff 87], [Tecuci 88] also derives explanations for inductive concept-learning. However, DISCIPLE only proposes conditions to be included in an explanation and asks the user to accept or reject them. In CLINT the user can but need not judge such partial explanations. Moreover, it is not possible to shift the bias in DISCIPLE because of a fixed language (cf. [Kodratoff 87],[Tecuci 88]). Also, in DISCIPLE it is less clear which concepts can be learned because DISCIPLE uses a *reduced* versionspace. This makes it difficult to decide *when* to shift the bias. A related difficulty with the reduced versionspace is the identification of the element of the G-set (cf. [Tecuci 88]). CLINT does not suffer from any of the above sketched problems because it searches specific to general. As such, CLINT is promising for use in LAS operating in a weak, incomplete or imperfect domain theory [Rajamoney 87]. Applying CLINT in imperfect or incomplete domains is feasible because CLINT is sound as it guarantees to identify the concepts in the limit (cf. [De Raedt 88a]), and uses an intelligent debugging method to retract all incorrect knowledge. The debugging method is based on Shapiro's work [Shapiro 82], but tuned towards learning, as all questions asked by the system are questions about the nature of an example.
Because of the questions to the user, asking whether an example belongs to the concept or not, our work on CLINT is also related to the work on experimentation [Sammut 86], [Subramian 86]. The way the questions are formulated is similar to Sammut's MARVIN, but MARVIN requires direct relevance. It is less clear which class of concepts can be learned with MARVIN. Subramian's method generates examples from the version-space. Since the versionspace of the languages presented here may be large her technique may give rise to combinatorial problems (the G-set may become very large). Our representation of the versionspace (the S-set and the set of negative examples) is more efficient for these languages. Since we do not use the G-set our technique is more general and a little less powerful as we cannot use the full power of factoring (cf. also [De Raedt 88a]).
Another line of related work are classical concept-learning approaches like [Michalski 83],[Sammut 86], where the examples specify some relations between objects in the examples, and where the system extends the descriptions of the example using predicates in the knowledge base. However, there the problem is less difficult as the system starts with an explanation and may generalize it [Sammut 86] or extend it [Michalski 83], while we do not start from any explanation at all. In this way, our work also resembles Buntine's PGA [Buntine 87]. PGA also copes with indirect relevance in a similar way as CLINT, but it cannot shift its bias.
Finally, our approach is also related to the work of [Utgoff 82],[Utgoff 86],[Ganascia 88] on bias and shifting the bias. Previous work on bias has concentrated on propositional calculus and is therefore rather limited. One of the main contributions of this paper is the introduction of series of languages, which are subsets of first order logic and which can be used for shifting the bias. Hence, the work of Utgoff is extended here.

Acknowledgements.

The reported work was partly done while the first author was visiting the Inference and Learning Group of Yves Kodratoff at the University of Paris-Sud. He benefited from discussions with Yves Kodratoff, Gheorghe Tecuci, Jean-Yves Lapique and Jean-Francois Puget. The visit was partially supported by the Belgian National Fund for Scientific Research by means of an IBM travel grant. The authors are also grateful to Gunther Sablon for his valuable comments on earlier versions of this paper and his help in preparing the final version.

References.

Angluin, D., Smith, C.S., Inductive inference : theory and methods, *Computing Surveys*, Vol. 15. No. 3, 1983, pp. 237-269.

Biermann, A., Fundamental mechanisms in machine learning and inductive inference, in : Bibel, W., Jorrand, P., (eds.), *Fundamentals of Artificial Intelligence*, (Springer-Verlag, Berlin-Heidelberg, 1986), pp. 133-170.

Buntine, W., Induction of Horn-Clauses : methods and the plausible generalization algorithm, *International Journal of Man-Machine Studies*, Vol. 26, No. 4, 1987, pp. 499-520.

DeJong, G., Mooney, R., Explanation based learning : an alternative view, *Machine Learning*, Vol. 1, No. 2., 1986, pp. 145-176.

De Raedt, L., Bruynooghe, M., On interactive concept-learning and assimilation, in : *Proceedings of the 3rd European Working Session On Learning*, Pitman, Glasgow, 1988, pp. 167-176.

De Raedt, L., Bruynooghe, M., Towards Friendly Concept-Learners, to appear in : *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, Detroit, 1989.

Dietterich, T.G., Michalski, R.S., A comparative view of selected methods for learning from examples, in : Michalski, R.S., Mitchell, T.M., Carbonell, J.G., (eds.), *Machine Learning : an artificial intelligence approach*, (Tioga publishing company, Palo Alto, 1983), pp. 39-81.

Ganascia, J.G., Improvement and refinement of the learning bias semantic, *Proceedings of the 8th European Conference on Artificial Intelligence*, Munich, 1988.

Genesereth, M.R., Nilsson, N.J., Logical foundations of Artificial Intelligence, (Morgan Kaufmann, Los Altos, 1987).

Hayes-Roth, F., Waterman, D., Lenat, D.B., Building Expert Systems, (Addison-Wesley, Reading Massachussets), 1983.

Kodratoff, Y., Is Artificial Intelligence a subfield of computer science or is Artificial Intelligence the science of explanations, in : Bratko, I., Lavrac, N., (eds.) ,*Progress in Machine Learning*, (Sigma Press, Bled, 1987), pp. 91-106.

Kodratoff, Y., Tecuci, G., Disciple : an interactive expert and learning system for weak theory domains, Research Report L.R.I., no. 373, 1987.

Kodratoff, Y., Introduction to Machine Learning, (Pitman, Paris, 1988).

Lloyd, J.W., Foundations of logic programming, (Springer Verlag, Berlin, 1984).

Michalski, R.S., A theory and methodology of inductive learning, in : Michalski, R.S., Mitchell, T.M., Carbonell, J.G., (eds.), *Machine Learning : an artificial intelligence approach*, (Tioga publishing company, Palo Alto, 1983), pp. 83-134.

Mitchell, T.M., Version Spaces : an approach to concept-learning, (Ph.D. Thesis, Stanford University, 1978).

Mitchell, T.M., The need for biases in learning generalizations, (Report. No. CBM-TR-117,

Department of Computer Science, Rutgers University, 1980).

Mitchell, T.M, Generalization as search, *Artificial Intelligence*, Vol. 18, No. 2., 1982, pp. 203-226.

Mitchell, T.M., Keller, R.M., Kedar-Cabelli, S.T., Explanation based generalization : a unifying view, *Machine Learning*, Vol. 1, No. 1, 1986, pp. 47-80.

Murray, K.S., Multiple convergence : an approach to disjunctive concept acquisition, in : *Proceedings of the tenth International Joint Conference on Artificial Intelligence*, Milano, 1987, pp. 297-300.

Quinlan, J.R., Induction of decision trees, *Machine Learning*, Vol. 1, No. 1, 1986.

Rajamoney, S., DeJong, G., The classification, detection and handling of imperfect theory problems, in : *Proceedings of the tenth International Joint Conference on Artificial Intelligence*, Milano, 1987, pp. 205-207.

Rendell, L., A general framework for induction and a study of selective induction, *Machine Learning*, Vol. 1, No. 2, 1986, pp. 177-226.

Russel, S.J., Analogical and Inductive reasoning, Ph. D. Thesis, 1986.

Sammut, C., Banerji, R., Learning concepts by asking questions, in : Michalski, R.S., Mitchell, T.M., Carbonell, J.G., (eds.), *Machine Learning : an artificial intelligence approach*, Vol. 2, (Morgan Kaufmann , Los Altos, 1986), pp. 167-192.

Shapiro, E.Y., Algorithmic program debugging, (The MIT Press, Cambridge, 1982).

Sterling, L., Shapiro, E.Y., The art of Prolog, (The MIT Press, Cambridge, 1986).

Subramian, D., Feigenbaum, J., Factorization in experiment generation, in : *Proceedings of the fifth National Conference On Artificial Intelligence*, Philadelphia, 1986, pp. 518-522.

Tecuci, G., DISCIPLE : A theory, methodology and system of expert knowledge acquisition, Thesis, Universite Paris-Sud, Orsay, 1988.

Utgoff, P.E., Mitchell, T.M., Acquisition of appropriate bias for concept learning, in : *Proceedings of the second National Conference On Artificial Intelligence*, Philadelphia, 1982, pp. 414-417.

Utgoff, P.E., Shift of bias for inductive concept-learning, in : Michalski, R.S., Mitchell, T.M., Carbonell, J.G., (eds.), *Machine Learning : an artificial intelligence approach*, Vol. 2, (Morgan Kaufmann, Los Altos, 1986), pp. 107-148.

A LEARNING EXPERT SYSTEM FOR PREVENTIVE MAINTENANCE

R.DUPAS P. MILLOT

LABORATOIRE D'AUTOMATIQUE INDUSTRIELLE ET HUMAINE

U.R.A. C.N.R.S. D0292

UNIVERSITE DE VALENCIENNES ET DU HAINAUT - CAMBRESIS

LE MONT HOUY
59326 VALENCIENNES CEDEX - FRANCE

A) INTRODUCTION

The Industrial Maintenance Data Bases enable the following up and the programming of all the activities of the maintenance department : technical, financial and managerial.

However, these Data Bases offer a problem of exploitation for the great amount of potentially stored information /WIEDERHOLD & AL 86/. Classical ways of inquiring and consulting data have proved their limits. Indeed they do not allow to meet the following problems:

- overload of information for the users of the Data Base.

- acquisition of knowledge from the Data Base.

More precisely the factory maintenance engineers are faced with the problem of exploiting the log of records concerning the equipment. They aim at extracting a strategy of maintenance with a view to optimizing the availability of the manufacturing process. To achieve this, they must be assisted by a decision aid system for predictive maintenance. This kind of maintenance consists in performing actions of preventive maintenance according to the actual state of the equipment and before a breakdown happens.

Statistical methods of analysis, and particularly the identification of the survival law brings a first element of answer. However it is necessary to complete this analysis with others means.

Our approach to these problems consists in the use of Machine Learning techniques /KODRATOFF 86/.

We aimed at automatically acquiring knowledge from the Data Base and exploiting it with a view to obtaining an aid to preventive maintenance. In the following part we will discuss the details of the knowledge acquiring system.

B) ACQUIRING A KNOWLEDGE BASE FROM THE DATA BASE

The problem is the transformation of crude data into knowledge offering a possibility of exploitation. The inductive learning systems are the best adapted for this purpose. Actually this kind of systems induce knowledge represented under the form of a decision tree or production rules, from examples. In our application the examples are made up of the values of diverse attributes taken from a table of the relational Data Base.

Our knowledge acquiring system is therefore based upon the use of an inductive learning technique (CHARADE system) associated with an empirical generalization method. We will first study the inductive module of the knowledge acquiring system.

1) Inductive module

1-1) The diverse inductive method

There are three main classes of inductive methods:

a) The algorithm family called Top Down Induction of Decision Tree (T.D.I.D.T.)

This family is based upon the algorithm ID3 /QUINLAN 86a/ and / QUINLAN 86b/. It allows to induce a decision tree for the classification of examples . The nodes of the tree represent the attributes and the branches of the tree, the values of these attributes. The basic idea is that a choice among attributes must be made according to the criterion of the gain of information wich must be optimum. Hence the drawing up of the tree proceeds by iterative specialization to obtain a good classification of examples.

There are two ways of improvement of the basic principle:

- the use of other criteria : for example statistical criteria for the treatment of noisy data /QUINLAN 87/.

- the translation of the decision tree into production rules for reason of intelligibility /CORLETT 83/ and /MANAGO 87/.

b) *The algorithm family AQ11 and its derivatives*

This family is based on the working out of a covering function that covers all the examples and that covers no counter-example /MICHALSKI & AL 80/. This method adapts the covering function by increment along the encountering of new examples until all examples are used up /REINKE & AL 88/.

c) Exploration of Hilbert' s cube : algorithm CHARADE

This method is based on the detection of regularities in the set of examples {E }/GANASCIA 87/. This is a two step process :

- the singling out of the sub-set {Ei} of {E} linked with the set of descriptor {di}, and so on for all the other descriptors and sets of descriptors. (a descriptor being a particular value within an attribute)

- the research of the set of descriptors shared in common by the examples forming the sub-set Ei : {dj}

So the rule {di} implies {dj} is induced.

The exploration of the whole space of descriptors is restrained by stating constraints of the process, which enables to adapt the set of rules to their finality. These constraints which will be seen in detail in paragraph 1-2-b, represent the true concept of intelligent learning.

1-2) Characteristics of our field of application and choice of a method

a)*Characteristics of our field of application*

The data feeding the Industrial Maintenance Data Base come from different origins:

- either a manual data acquisition by a maintenance operator.

- or an automatic data acquisition from sensors set on machines or monitoring the process.

The automatic data acquisition warrants the reliability of the data fed into the Data Base. On the other hand manually acquired data may be incorrect, because they are bound to the reliability of the human operator. Analyses or deductions based upon these unreliable data may mislead the engineer in charge of maintenance.

Though the general trend is to replace manual data acquisition by automatic ones, a number of data can not be acquired automatically (for instance: the description of a sign of fault or diagnosis). This explains why an inductive module coping with noisy data is necessary.

Another point to take into account in the choice of an inductive module is the permanent updating of the Data Base. That prompts us to choose an inductive module that enables incremental learning.

b) *choice of a method*

We have chosen the CHARADE system because it has the two necessary conditions required for our application :

- the treatment of noisy data which is realized through a specific constraint of the exploration (see under).

- the incremental knowledge acquisition working in the following way: when it happens that a new example E contradicts the system of rules, the rules originating this contradiction are deleted.

On the contrary , the T.D.I.D.T. algorithm family does not allow easy incremental learning and the AQ11 algorithm family does not permit the treament of noisy data.

Moreover CHARADE system offers the following extra advantages /GANASCIA 88/:

- It produces a true system of rules , some of which being linked in chains. (A -> B and B -> C) and including no redundancy.

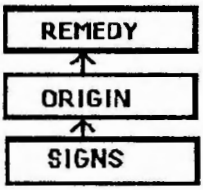
- It is not exclusively aimed at diagnosis problem, as ID3 and AQ11 are.

The constraints we have chosen to apply to explore the space of description are determined by the characteristics of our application. Particularly:

- minimal number of examples covered by the premise of a rule. This constraint permits to resist noisy data.

- possibility of inducing a system of rules aimed at classification or diagnosis .(in a classification system , the description of a class is exclusive; in a diagnosis system the description of a diagnosis is not exclusive)

- possibility of inducing a system of rules with a predefined structure. For example, in the field of preventive maintenance, the structure of the ad hoc system of rules is :



- limitation of the number of descriptors for each premise. This constraint enables to reduce the time of exploration, mostly in the first attempts of induction.

It is worth mentioning that we have limited the acquisition of rules to logical ones. We have not tried to work out approximate rules as defined in CHARADE system at this stage of

development of the project.This possibility is interesting in the context of our application because it encompasses noisy data but it makes the use of the system of rules more difficult.

In the next part we will study the generalization module which must be applied to the examples before acquiring rules with CHARADE.

2) Generalization module

In the CHARADE system the generalization of examples is obtained during the completion stage /GANASCIA 87/ and /VRAIN & AL 86/. This stage is based upon the use of properties of the example attributes. It presupposes that the attributes used for the representation of examples have been described through a set of properties and axioms.

Within the scope of our particular application the implementation of this stage of example completion that implies the use of an object oriented language, cannot be considered. As a matter of fact the finality of our knowledge acquisition tool is to be integrated into the Data Base. That is why we have utilized a simplified generalization mechanism that does not require a frame based language. This generalization stage is based upon a simple generalization rule in association with an elementary domain background knowledge contained in the Data Base.

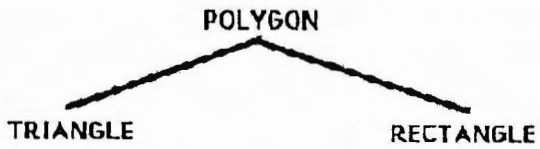
2-1) Generalization rule

The generalization rule we use is part of the set of empirical generalization rules defined by Michalski : it is the rule of climbing generalization tree /MICHALSKI 83/.

a) climbing generalization tree

We have chosen this rule because it is suited to the type of data in our Data Base (see paragraph 2-2). To apply this rule, the examples must have at least one attribute of hierarchical type.

Suppose the attribute "form" is associated with the following hierarchy:



If we have the two examples :

E1 : [form = triangle] et [rest of the description of the example :D]

E2 : [form = rectangle] et [rest of the description of the example :D]

they can be generalized in Eg.

Eg : [form = polygon] et [rest of the description of the example :D]

provided that, the two examples E1, E2 differ only by the value of the descriptor "form".

b) ***domain background knowledge***

This has been described as a major part of an inductive learning system by Michalski /MICHALSKI 83/. In our application this background knowledge consists in attributes hierarchically linked.

Hereafter, two examples :

- the hierarchical description of the machine structure with four levels.

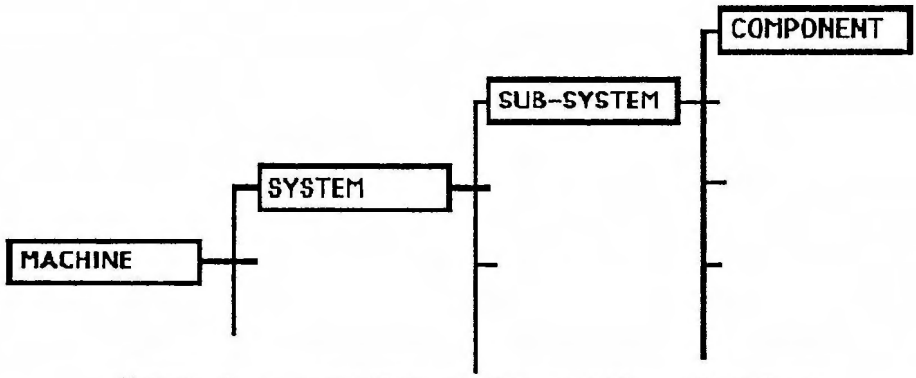


Figure 1: description of the machine structure

example:

Machine : Lathe T1
System : Rotating part
Sub-System : Electrical engine
Component : Rotor

- the hierarchical description of the breakdown structure with tree level.

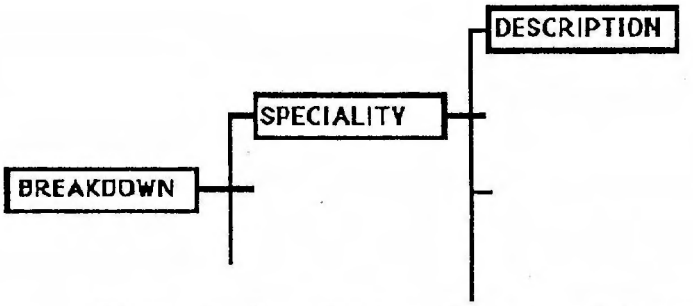


Figure 2: description of the breakdown structure

example:

Breakdown	: P1
Speciality	:Mechanic
Description	:Vibration

The stage of acquisition of production rules being described, we will now study how to exploit these rules.

C) LEARNING EXPERT SYSTEM FOR PREVENTIVE MAINTENANCE

1) The component parts of the Expert System

The component parts of the expert system we developed are represented on figure 3. Since we have already described the learning algorithm and the characteristics of the Knowledge Base it induces, we have now to describe the Knowledge Base of preventive maintenance and the expert system building tool we use.

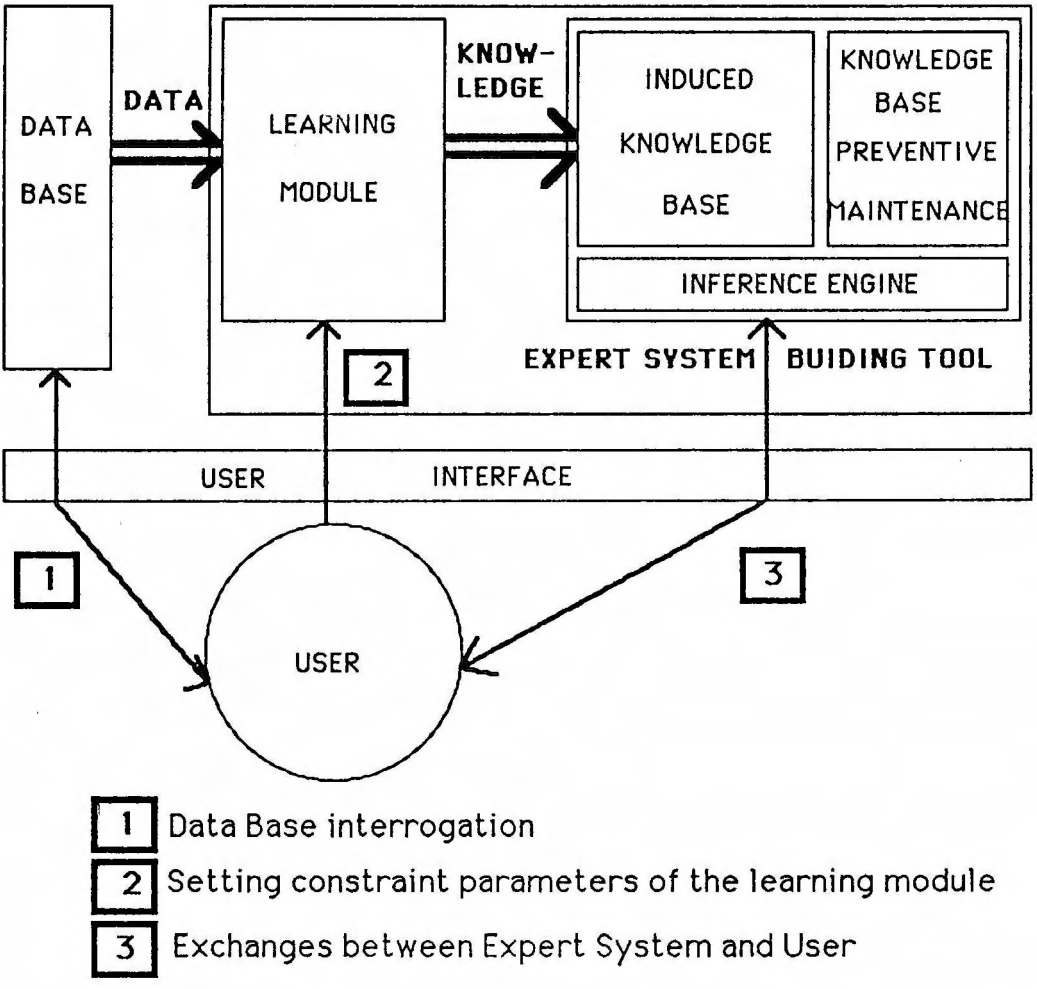


Figure 3: the component parts of the Learning Expert System

1-1) knowledge base of preventive maintenance

To finalize an aid system useful in preventive maintenance, it has been necessary to associate to the base of rules induced by CHARADE a knowledge base of preventive maintenance. That is because the system of rules induced by CHARADE is built with the attributes of the Data Base exclusively . These rules have to be completed and interpreted to produce valuable conclusions in the practical field of preventive maintenance. These additional rules are general practice rules of maintenance that do not depend on any particular application.
Here is an example :

If fault = design or fault = making or fault = assembling

Then call the manufacturer

To date we have at our disposal about ten rules of this kind. Their number will increase along with the development of this project.

1-2) expert system building tool

The expert system building tool we use enables the managing of production rules issued in propositional logic. It combines the rules induced by the learning algorithm and the rules drawing on maintenance expertise.

2- Running the expert system

It is a two step process: the first step consists in inducing the Knowledge Base from the Data Base, and the second step is the exploitation of the acquired knowledge.

2-1 Induction

During this step the user has :

- first, to define a subset of attributes chosen from the Data Base to build a data table to be analysed by the learning system.

- second to set the values of the different constraints required by the algorithm CHARADE. This choice depends on the expected properties of the acquired set of rules (see § B-1-2-b). It may be useful to repeat this task in order to adjust the constraints by approximation according to the time of computation and to a rough assessment of the acquired rules.

2-2 Exploitation

This step aims at obtaining conclusions of preventive maintenance starting from a predetermined category of facts. For example we may have chosen to analyse the diverse types of equipments or the characteristics of the teams of maintenance.

This implies:

- forward chaining with the system of rules induced by CHARADE starting from the initial facts. It gives an intermediary conclusion.

- chaining this intermediary conclusion with a rule of preventive maintenance.

Once this has been tried, if nothing valuable has been deduced it may be necessary to go back to the previous step of induction in order to change again the constraints of the algorithm CHARADE.

3) Preliminary assessment of this approach

We have worked on a set of 40 examples described with the help of 8 attributes with 4 values at most for each of them. This oversimplified study has already shown the following points:

Positive points
This approach aims at giving a solution to the problem of interpretation and filtering of irrelevant rules generated by a learning algorithm. The filtering is worked out, on one hand thanks to the constraint parameters of CHARADE, and on the other hand thanks to the Knowledge Base of preventive maintenance whose objective is to orient the deductions produced by the system towards a conclusion meaningful in the field. This can be seen as an improvement on the technique described as the manual edition of induced rules as recommended by Ruberg /88/.

Negative points
This method necessitates to be somehow familiar with the constraint parameters to be used in CHARADE.
Then, the degree of relevance of the study realised depends on the accuracy of the Knowledge Base of preventive maintenance.
Finally, the efficacy of the results may vary greatly with the category of facts chosen as the starting points.

D) CONCLUSION

This paper presents an original way of exploiting an Industrial Maintenance Data Base by a learning expert system for preventive maintenance.

This approach has two main advantages :

- the knowledge acquisition process is a useful way of challenging the classical statistical studies.

- coupling the use of the acquired Knowledge Base and of the preventive maintenance Knowledge Base by the means of an expert system tool, permits a better utilisation of the acquired knowledge.

The future desirable improvements to be achieved are:

- continuing with the assessment of this approach in two ways: by testing more important sets of examples and by refining the Knowledge Base of preventive maintenance with experts in this field.

- as far as the algorithm CHARADE is concerned, implementing the acquisition of approximative rules and studying the implementation of incremental learning.

- studying the use of conceptual clustering techniques in order to cluster data before utilizing the system of rule induction.

This system implemented on a P.C A.T.
The Data Base is developed with the Data Base Management System ORACLE. The knowledge acquisition algorithm coupled with the Data Base is written in C language. The expert system building tool is Intelligence Service II.

BIBLIOGRAPHY

CORLETT, 83. Explaining induced decision trees. Proceeding Expert System 83. pp 136-142.

GANASCIA, 88. Charade: une sémantique cognitive pour les heuristiques d'apprentissage. The 8th international worshop on expert systems and their applications : Avignon 88. PP 567-585.

GANASCIA, 87. AGAPE et CHARADE : deux mécanismes d'apprentissage symbolique appliqués à la construction de bases de connaissances , thèse d'état, Université Paris-Sud, mai 87.

KODRATOFF, 86. Leçons d'Apprentissage Symbolique Automatique. Cepadues-Editions 86.

MANAGO, 87. Journées "Symbolique Numérique "pour l'apprentissage de connaissances à partir de données. Université Paris Dauphine, décembre 87.

MICHALSKI, 83. A theory and methodology of inductive learning. In Machine Learning, an Artificial Intelligence Approach. Tioga publishing compagny, Palo Alto, California, 83. pp 83-134.

MICHALSKI and R.L. CHILAUSKY , 80. Learning by being told and learning from examples : an experimental comparison of 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 4 (2), 80. pp 125-160.

QUINLAN, 87. Simplifying decision trees. International Journal Man Machine Studies 87. Vol 27. PP 221-234.

QUINLAN, 86 a. Introductory readings in Expert Systems. D.Michie Ed., 86.

QUINLAN, 86 b. Induction of decision trees. In Machine Learning, an Artificial Intelligence Approach. Tioga publishing compagny, Palo Alto, California, 83. pp 81-106.

RUBERG & AL 88. Building and Maintening a Diagnostic Rule-Base. 3rd AAAI-sponsored Knowledge Acquisition for Knowledge-Based Systems Worshop Banff, Canada, November, 1988.

REINKE, R.S. MICHALSKI, 88. Incremental learning of concept descriptions : a method and experimental results. Logic and the acquisition of knowledge. Machine Intelligence J.E. Hayes, D. Michie and J. Richards Ed., 88. pp 263-288.

VRAIN & AL, 86. "AGAPE : an algorithm that learns from similarities". Proceedings of the European Working Session on Learning. Internal papre LRI, Université Paris-sud n° 258, Orsay 86.

WIEDERHOLD, M.G. WALKER, R.L. BLUM and S.M. DOWNS, 86. Acquisition of knowledge from data. Proceedings of the A.C.M. Sigart International Symposium on methodologies for Intelligent Systems. Knoxville 86. pp 74-84.

A Learning System for INFERNO Inference Networks

José Carlos González Cristóbal
Francisco Alcaraz Ramiro
Departamento de Ingeniería Telemática
E.T.S.I. Telecomunicación
Universidad Politécnica de Madrid (Spain)

Address: E.T.S.I. Telecomunicacion
Ciudad Universitaria
28040-Madrid (Spain)
Telephone: +34 1 4495700
FAX: +34 1 2432077
E-mail: jgonzalez@dit.upm.es
jgonzalez@goya.uucp

A Learning System for INFERNO Inference Networks *

José Carlos González
Francisco Alcaraz
Departamento de Ingeniería Telemática
Universidad Politécnica de Madrid

Abstract

INFERNO, as stated by J.R. Quinlan, its author, is a cautious approach to uncertain inference. It is a model for inference networks based on a strict application of probability theory. Lower and upper bounds are used to characterize the probability of every proposition (node) in the inference net and no assumption is imposed on the joint probability distribution of propositions. Deciding the topology of a model for a specific application domain is easy. However some difficulties arise to characterize the strength of links between nodes. We have designed and implemented an inductive tool to learn incrementally from examples the parameters that fix the behaviour of this kind of networks.

1 Introduction

It should be expected that learning systems are mainly applied to domains where knowledge is uncertain, inexact or incomplete. The reason is clear: it is supposed that these domains are specially complex for humans; therefore, any tool intended to reduce this complexity (variety, in the cybernetic

*This research has been supported by the spanish bank BANCO DE BILBAO VIZCAYA.

sense) would be greatly appreciated. This argument should be sufficient to support the necessity of a strong relation between approximate reasoning and learning techniques. However, reality is very different and both communities remain far apart.

The purpose of this paper is to show how the INFERNO system works and how it has been possible to build a learning tool able to characterize the parameters of this kind of inference networks from examples [1].

In a well-known paper [9], Quinlan presented a new approach to uncertain inference. First he analyzed how key aspects in inference networks, such as knowledge representation, assumptions and control in propagation of information, and treatment of inconsistencies, were addressed by such well established methods as Bayes theorem, certainty theory and evidence theory.

This analysis led him to state the basis of his INFERNO system:

1. The system should not depend on assumptions on probability distribution of propositions.

2. It should be possible to assert relations between propositions, e.g. independence, if known.

3. There should be no distinction between propositions, e.g. evidence versus hypothesis, allowing to assert information about any node in the net.

4. Inconsistent information should be detected, showing ways to amend this situation.

Since its publication, INFERNO has been considered a theoretically very sound approach to uncertainty, deserving its inclusion in textbooks about knowledge based systems [3] and reviews of existing methodologies for managing uncertainty [7,10]. Even though there is an appreciable quantity of work in the literature about the model [2,4,5,6,8], there is nothing about its use enhanced with learning capabilities. This fact, and our interest in building real expert systems for the bank supporting the project, persuaded us that some additional work should be done:

- A learning system was indispensable to help the users to asses the strength of causal relations.
- The system should be based on the rectification mechanism integrated in the INFERNO system.

Next, after a brief description of the methodology (section 2), we will address the objectives, algorithm and implementation of our learning system (section 3).

2 Description of the INFERNO system

2.1 Knowledge representation

In INFERNO, belief in any proposition A, node of the inference net, is characterized by two values, $t(A), f(A)$, such that its probability lies in the real interval $[t(A), 1-f(A)]$, that is,

$$p(A) \geq t(A), \qquad p(\bar{A}) \geq f(A)$$

In the other side, relations between propositions (see table 1) fall into three classes:

- Strength dependencies (*enables, inhibits, requires, unless*), characterized by parameters that are interpreted as lower bounds to conditional probabilities between propositions.
- Logical dependencies (*negates, conjoins and disjoins*). In addition to the simple conjunction and disjunction relations that involve no assumption on dependencies between propositions $\{B_1, \ldots, B_n\}$, other relations permit assertions of independence or mutual exclusion between them. This meets 1st and 2nd items in the INFERNO declaration of intentions as stated in the previous section.
- Mutual exclusion.

Relation	*Interpretation*
B enables A with strength e	$p(A\|B) \geq e$
B inhibits A with strength i	$p(\bar{A}\|B) \geq i$
A requires B with strength r	$p(\bar{A}\|\bar{B}) \geq r$
A unless B with strength u	$p(A\|\bar{B}) \geq u$
A negates B	$A = \bar{B}$
A conjoins $\{B_1, \ldots, B_n\}$	$A = B_1 \wedge \cdots \wedge B_n$
A disjoins $\{B_1, \ldots, B_n\}$	$A = B_1 \vee \cdots \vee B_n$
A conjoins-independent $\{B_1, \ldots, B_n\}$	$A = B_1 \wedge \cdots \wedge B_n$ $\forall i \neq j,\ p(B_i \wedge B_j) = p(B_i)p(B_j)$
A disjoins-independent $\{B_1, \ldots, B_n\}$	$A = B_1 \vee \cdots \vee B_n$ $\forall i \neq j,\ p(B_i \wedge B_j) = p(B_i)p(B_j)$
A disjoins-exclusive $\{B_1, \ldots, B_n\}$	$A = B_1 \vee \cdots \vee B_n$ $\forall i \neq j,\ p(B_i \wedge B_j) = 0$
$\{B_1, \ldots, B_n\}$ mutually exclusive	$\forall i \neq j,\ p(B_i \wedge B_j) = 0$

Table 1: Relations in INFERNO and their interpretation (from [9])

Each one of these relations has associated a group of propagation constraints. Constraints impose conditions on values $t(A), f(A)$ that must be always satisfied by every proposition A involved in the relation. As an example, these are the INFERNO constraints for *enables* relations.

$$t(A) \geq t(B)e \qquad\qquad f(B) \geq 1 - \frac{1 - f(A)}{e}$$

Although relations are oriented, information can proceed in both senses, as can be seen in these constraints. This fact permits the achievement of point 3 expressed in section 1. Constraints can be demonstrated (see [9] for proofs) from their interpretation through basic probability theory.

2.2 Propagation

Values $t(A) = f(A) = 0$ are assigned initially to every node in the network, showing absolute ignorance of probabilities of propositions. Each time new information arrives at the net in form of new values t or f for any node A, higher than previously known, relations in which this bound takes part must be activated. This activation means calculation of the right side expression of the respective constraint. If the calculated value is higher than the present one, former value must be replaced and the propagation continues until equilibrium is reached.

2.3 Consistency and rectification

In INFERNO there is a way to detect and correct inconsistencies between informations given to or deduced by the network (meeting the last point discussed in section 1). When it happens that, for some node A, $t(A) + f(A) > 1$, it means that at least one of the probability bounds $t(A), f(A)$ must be wrong. Causes that may lead to this situation are:

- wrong information supplied by the user.
- wrong net design by experts or knowledge engineers.

INFERNO makes use of a rectification algorithm to obtain alternate solutions ensuring the overall consistency of the net. These solutions (called rectifications by Quinlan) involve two types of changes:

1. Changes in user supplied bounds.
2. Changes in the strength of relations.

The original rectification algorithm works as follows:

1. Bounds of inconsistent nodes are modified making $t'(A) = 1 - f(A)$ or $f'(A) = 1 - t(A)$
2. As these bounds were calculated from previous ones through propagation constraints, backward chaining must be used to propagate the selected modification. Two situations are considered:
   - Strength relations, where inconsistency can be eliminated by reduction of the correspondent strength.
   - In any case, it is always possible to reduce some of the bounds of ancestor nodes to achieve consistency, and the whole process must begin again.

Backward chaining is achieved through application of reverse propagation constraints. As an example, these are the reverse constraints for *enables* relations:

$$t'(B) \leq t(A)/e \qquad f'(A) \leq 1 - (1 - f(B))e$$

When the algorithm is applied to every inconsistent node, a set of rectifications is obtained. Each rectification is a set of changes in probability bounds and/or strengths that make the network consistent.

Finally, it is necessary to use some fixed criterion to select the best rectification. The criterion used by Quinlan is based on the concept of *reluctance*, a measure of the change imposed by the acceptance of some particular rectification:

$$\mathcal{R}_j = \sum_m \delta s_m + \sum_N (\delta t(N) + \delta f(N))$$

where δs_m is the strength reduction for a particular relation m (enables, inhibits, requires or unless), and $\delta t(N)$, $\delta f(N)$ are the reduction of the values t and f for any particular node N.

3 The learning system

3.1 Objectives

Our experience building expert systems with INFERNO has shown that deciding the topology of a network for specific application domains is not very difficult. However, human experts find very hard the characterization of every strength link by a number. Therefore we have designed and implemented an inductive tool able to learn these parameters from examples in an incremental way.

Inductive learning systems acquire knowledge through examples supplied by human experts. Examples consist of a set of known facts (input information) and a set of results or conclusions inferred by experts. Incremental learning makes considering examples one by one possible. Whenever a new case is considered the system produces results that must be compared with those suggested by experts. As a consequence of this comparison the artificial system must be able to modify its own knowledge base to keep its behaviour as close as possible to human experts (figure 1).

Our learning system takes sets of facts or conclusions consisting of lists of nodes besides their t and f values. Initially every parameter (relation strength) in the net is assigned the value 1. Then, the given set of facts is propagated according to the principles shown in section 2.2 until all constraints are satisfied. If t or f values reached during propagation for any node in the conclusions set is higher than the maximum value given in this set, then the rectification algorithm will reduce it to the maximum permitted value (figure 2). This reduction is accomplished by weakening the strength of some links.

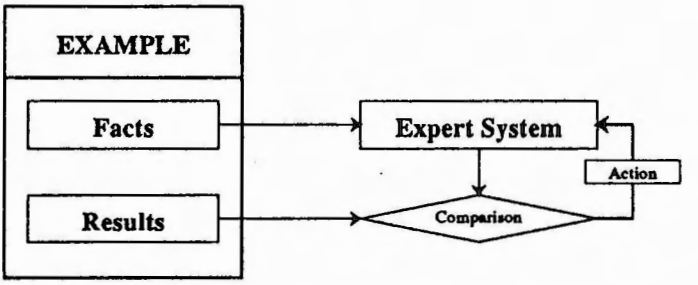


Figure 1: An incremental inductive learning system

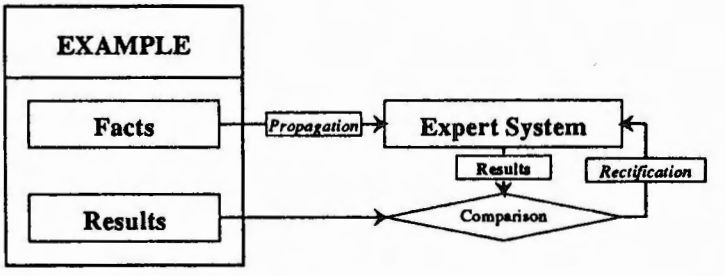


Figure 2: The INFERNO learning system

The main advantage of this method is that the rectification algorithm will find the best set of rectifications to adapt the net to the behaviour shown by human decision-makers.

3.2 Modifying the rectification algorithm

Our learning system accomplishes some interesting modifications in the rectification algorithm presented by Quinlan [9] exhibiting a significant increase of flexibility and power:

1. Completeness of rectifications

   During backward chaining, the original rectification algorithm checked the trace of each inconsistent node, where the origin of actual t and f values was stored. However we have noticed that sometimes the algorithm was unable to recover the net from its inconsistent initial state. This is due to the fact that any node can be made inconsistent through several affecting relations. So, backward chaining must be done by checking every node related with those reported as inconsistent ones.

2. Side-inconsistencies

   As it has been remarked above, the process of tuning the network is similar to the process of correcting inconsistencies. So, we have called side inconsistencies those originated through incompatibility between experts assessment and conclusions inferred from the actual state of the network. This new type of inconsistency implies more flexibility using the INFERNO system.

3. Optimized search for valid rectifications

   Rectifications can't be computed separately for each node. Better solutions are found when they are explored in an orderly way, starting each rectification from the situation reached after the rectification of previous nodes.

3.3 Reluctance weighting

When the modified rectification algorithm is used for learning purposes, it is necessary to take into account that the network is not as reliable when no example has been processed, as it is after considering a great number of them.

The learning algorithm must reduce the initial strength of some dependencies, but these values (that represent the adaptation of the net to the training set) must be protected from inaccurate or incorrect data that could lead the system towards undue and irrecoverable reductions.

Thus, this important observation has been considered in our system to modify the computation of reluctance. When the number of previously processed examples increases, rectifications involving changes in strength relations must have higher comparative weights. The new formula shows a weighting factor dependent on the number of examples processed:

$$\mathcal{R}_j^i = \sum_m \delta s_m + K_i \sum_N (\delta t(N) + \delta f(N))$$

where, $\mathcal{R}_j^i$ = Reluctance of the rectification j
when example i is being processed
$K_i = i_o/i$ Weighting factor for example i
i_o = Number of examples such that changes in strengths are weighted the same as changes in user supplied results

Examples must be ordered according to generality and reliability, with over-specific or doubtful cases placed at the end of the list. Moreover, constant i_o must be selected depending on the whole number of available examples i_t. In our experiments, we have used the criterion roughly represented in figure 3.

3.4 The learning algorithm

This is the final learning algorithm, expressed as Pascal-like pseudocode, after introducing the changes mentioned in previous sections:

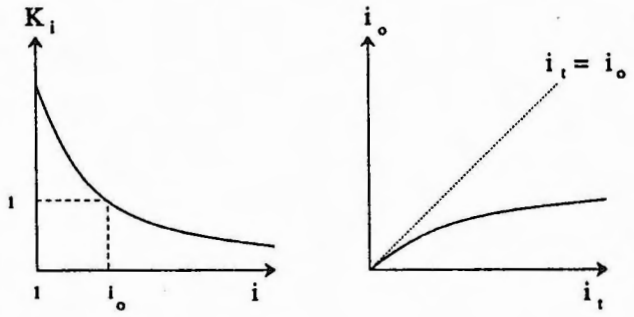


Figure 3: Selection of values for K and i_o

```
FOR r:=1 TO m DO strength[r]:=1; {Initialize strength relations}
i:=0;                            {Number of examples processed}
WHILE i<imax DO BEGIN
    i:=i+1;
    FOR N:=1 TO Nmax DO BEGIN    {Initialize t and f values}
        t[N]:=0;
        f[N]:=0;
    END;
    Propagate_Facts;      {Propagation                          }
    Check_Results;        {Production of side inconsistencies }
    Rectificate;          {Rectification                        }
    Accept_best_rectif;   {Selection of the best rectification}
END;
```

Figure 4 shows a very simplified network for credit assignment. These are the variables included in the model:

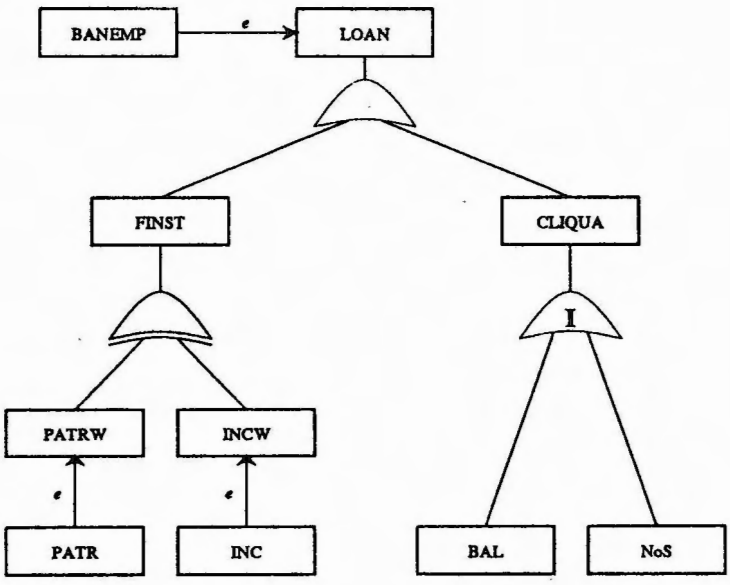


Figure 4: A simple example

Loan:	Loan Assignment
FinSt:	Finantial Situation
PatW:	Weighted Patrimony
IncW:	Weighted Income
Patr:	Client Patrimony
Inc:	Client Income
BanEmp:	Client worker of the bank
CliQua:	Client quality
Bal:	Account balance
NoS:	Bank services used by the client

Values obtained for enable relations with a training set of 20 examples:

BanEmp $\longrightarrow$ Loan with strength $e = 0.52$
Patr $\longrightarrow$ PatrW with strength $e = 0.39$
Inc $\longrightarrow$ IncW with strength $e = 0.61$

4 Conclusions

In this paper we have presented an inductive learning system for INFERNO inference networks. Human experts must supply the system with the basic structure of the net and a set of examples consisting of ground facts and their own assessment of truth values for whatever remaining propositions. The learning system considers incrementally every example and modifies the strength of causal dependencies if the knowledge supplied by experts is incompatible with the conclusions inferred by the network. Modifications of the net are weighted against modifications of values supplied by human experts depending on the experience gained by the system. This experience is related with the number of examples previously processed.

At this moment the system is being tested with some models for credit card assignment and provisional results are promising. However, some work remains to be done in three different areas:

- Assessing the usefulness of the INFERNO system as a methodology for knowledge representation.
- Testing the learning system with additional real size expert systems.
- Studying the feasibility of learning network structure rather than network parameters.

5 Acknowledgments

Our Department is indebted to Emilio Rincón and Miguel A. Fernández of the spanish bank Banco de Bilbao-Vizcaya for their unconditional support for the project.

We wish to thank the graduate and undergraduate students E. Rodríguez, J.L. Blázquez, E. Herranz, T. Domínguez, J.M. Goñi and S. Uriel for their participation in the implementation of different versions of the INFERNO system.

References

[1] F. Alcaraz. *Sistema de Aprendizaje en Redes de Inferencia.* Master's thesis, E.T.S.I. Telecomunicación (Universidad Politécnica de Madrid, Spain), 1988.

[2] A. Blake. Probabilistic inference by linear optimization. In A.G. Cohn and J.R. Thomas, editors, *Artificial Intelligence and its Applications. AISB-85, Conference on AI and its Applications*, pages 139–151, Society for the Study of AI and Simulation of Behaviour, John Wiley, 1986.

[3] R.A. Frost. *Introduction to Knowledge Based Systems.* Collins, 1986.

[4] A. Gammerman and N. Creaney. Modelling of uncertainty in expert systems. In *Proceedigns of the Second International Expert Systems Conference*, pages 265–273, Learned Information (Europe), 1986.

[5] X. Liu and A. Gammerman. A hybrid approach to deductive uncertain inference. *International Journal on Man-Machine Studies*, 28:1–11, 1988.

[6] X. Liu and A. Gammerman. On the validity and applicability of the INFERNO system. In M.A. Brammer, editor, *Research and Development in Expert Systems III*, pages 47–56, British Computer Society, Specialist Group on Expert Systems, Cambridge University Press, 1987.

[7] D. Pang, J. Bigham, and E.H. Mamdani. Reasoning with uncertain information. *IEE Proceedings, Pt.D*, 134(4):231–237, 1987.

[8] J.R. Quinlan. Consistency and plausible reasoning. In A. Bundy, editor, *Proceedings of the Eighth International Conference on Artificial Intelligence*, pages 137–144, William Kaufmann, Inc., 1983.

[9] J.R. Quinlan. INFERNO: a cautious approach to uncertain inference. *The Computer Journal*, 26(3):255–269, 1983.

[10] D.J. Spiegelhalter. A statistical view of uncertainty in expert systems. In W.A. Gale, editor, *Artificial Intelligence and Statistics*, Addison-Wesley, 1986.

Acquiring Knowledge about A Relation between Concepts

Ken-ichi HANDA and Shun ISHIZAKI
Electrotechnical Laboratory
1-1-4, Umezono, Tsukuba, JAPAN 305
E-mail: handa%etl.jp@relay.cs.net

Abstract

A method for acquiring knowledge about a relation between concepts is described. This method searchs a knowledge base for insufficient parts and acquires indispensable information through an interaction with a human. 5 different mechanisms of knowledge acquition are proposed and integrated in a system. While taking importance of concepts into account, the system created a realistic knowledge base from ambiguous information obtained from newspaper articles.

1 Introduction

Automatic concept learning is a big topic in the field of machine learning. A variety of methods have been proposed in this field. Some of them are based on analyzing positive/negative examples of a goal concept (e.g., [Winston 72]), others on comparing simple attributes of examples (e.g., [Michalski 80]).

However, to treat "real world problems", those methods still have the following problems. First, there's no teacher who carefully select or order input data for a system to learn a specific concept. Second, it's difficult to let the system know in advance what to learn. Third, data structures given to the system is so complicated that simple attribute-value comparison-based methods do not work. And last, "generalization/specialization", which is the only mechanism used in those methods (e.g., [Lebowitz 86]), is necessary but not sufficient to make the most of the opportunities to acquire knowledge.

In this paper, we propose "acquisition of a relation between concepts" as one form of concept learning. The method refine knowledge about a concept incrementally.

Each time new knowledge is given, the system searches for an insufficient part of the knowledge base, decides what knowledge is important for the knowledge base, and then try to get more knowledge from a human to fill up the shortage. In this process, judgment of importance is very crucial to decrease a humans teaching load. We adopted a method described in [Handa 88] to calculate an importance value of a concept.

Since knowledge acquisition in our system proceeds incrementally, the system provides a useful supporting tool for building a large scale knowledge base.

First, we investigate what kind of information an inter-concept relation adds to the knowledge base. This provides a logical basis for further discussion.

Next, we discuss the meaning of acquiring such information from the viewpoint of concept learning and propose five knowledge acquisition mechanisms, which are named *generalization, specialization, exclusion, categorization,* and *diplomacy.*

Finally, we describe the precise process of our system, how to find shortage of knowledge, how to decide what is important for the system, and how to ask to a human for the necessary knowledge.

2 Information of relation between concepts

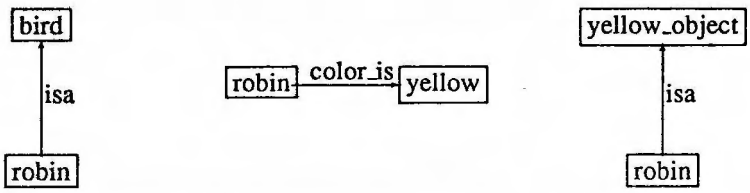
In a semantic network based knowledge base, a concept is represented by a node and the semantics of the concept defined by relations (links) the concept has with the other concepts. Here after, we treat of the semantic network only case of relations between two concepts (i.e. binary relations) existing.

Concepts are generally classified into *class* and *instance*, and a minimum unit of knowledge in such a knowledge base is a triplet of two concepts and a relation between them. We will discuss the information involved in such a unit according to the constraint on a type of *instance* in the knowledge base.

For instance, a relation between a *class* and an *instance* (generally called as "instance_of" relation) has the information that there surely exists the *instance.*

Information in a hierarchical relation between two *classes* can also be reduced to existence of *instances.* Figure 1.a shows a simple hierarchy between two *classes* "robin" and "bird". This relation has the information that there's no *instance* of "robin" which is not an *instance* of "bird". In other words, every *instance* of "robin" is necessarily an *instance* of "bird".

As for non-hierarchical relation between *classes*, we need a slight reorganization of the network for further discussion. Let's think about a relation shown in Figure 1.b. It is meaningless to argue about an instance of "yellow". But, if we translate the network into



a. hierarchical relation b. non-hierarchical relation c. translated relation

Figure 1: Relation between *classes* in semantic network

type of *instance* x	$A(x) \wedge B(x)$	$A(x) \wedge \neg B(x)$	$\neg A(x) \wedge B(x)$	$\neg A(x) \wedge \neg B(x)$
world1	t	t	t	t
world2	t	t	t	f
world3	t	t	f	t
world4	t	t	f	f
world5	t	f	t	t
world6	t	f	t	f
world7	t	f	f	t
world8	t	f	f	f
world9	f	t	t	t
world10	f	t	t	f
world11	f	t	f	t
world12	f	t	f	f
world13	f	f	t	t
world14	f	f	t	f
world15	f	f	f	t
world16	f	f	f	f

Table 1: Possible world considering two *classes* I

Figure 1.c, the same kind of discussion makes sense: we can say that there's no *instance* of "robin" which is not an *instance* of "yellow_object". "yellow_object" is a generic concept of objects that are yellow.

To make discussion more formal, we introduce predicate logic. A predicate represents a *class* and a predicate variable represents an *instance*. For example, $A(x)$ means that x is an *instance* of *class* A.

Now we show what kind of world we should take into account according to various types of relations between two *classes*. These worlds differ in potential types of *instances*. Being concerned with two *classes*, A and B, there exist 4 different types of *instance* x: $A(x) \wedge B(x)$, $A(x) \wedge \neg B(x)$, $\neg A(x) \wedge B(x)$, and $\neg A(x) \wedge \neg B(x)$. And, depending on the kind of those *instances* alone, worlds are classified into 16 different ones as shown in Table 1.

A symbol "f" in each column of the table means there does not exist that kind of *instance*, "t" means that there may exist that kind of *instance*. For instance, the *instance* x in "world12" is necessarily $(A(x) \wedge \neg B(x))$. The more column "f" a line for a world has, the more information the world has.

However, we should leave some of these worlds out of consideration because they put too many restrictions to a type of *instance*. In "world15", for example, both A and B have no *instances*, which is, however, a very strange situation in a realistic knowledge base.

After excluding those nonsense worlds from the table, we get Table 2, in which 5 possible worlds remain. In the rightmost column are shown a relation of A and B in that world. Restriction to a type of *instance* is the strongest in "world7" and the weakest in "world1". In other words, the "world7" has the largest information about *instances* and "world1"

type of *instance* x	$A(x)$ $\wedge$ $B(x)$	$A(x)$ $\wedge$ $\neg B(x)$	$\neg A(x)$ $\wedge$ $B(x)$	$\neg A(x)$ $\wedge$ $\neg B(x)$	information about *instance*	
world1	t	t	t	t		no relation
world3	t	t	f	t	$\neg(\exists x(A(x) \wedge \neg B(x)))$	$A \subset B$
world5	t	f	t	t	$\neg(\exists x(\neg A(x) \wedge B(x)))$	$A \supset B$
world7	t	f	f	t	$\neg(\exists x(A(x) \wedge \neg B(x)))$ $\wedge\neg(\exists x(\neg A(x) \wedge B(x)))$	$A \equiv B$
world9	f	t	t	t	$\neg(\exists x(A(x) \wedge B(x)))$	$A \oplus B$

Table 2: Possible world considering two *classes* II

$$\boxed{A \equiv B} > \boxed{A \subset B} > \boxed{\text{no relation}}$$

$$\boxed{A \equiv B} > \boxed{A \supset B} > \boxed{\text{no relation}}$$

$$\boxed{A \oplus B} > \boxed{\text{no relation}}$$

Figure 2: Comparison of amount of information

has no such information. Figure 2 shows comparison of amount of information brought by those relations. As a matter of fact, however, we should also exclude "world7" because different concepts should represent different meanings in a general knowledge base. So now, the possible relation we can give to the knowledge base is restricted to one of "$\subset$", "$\supset$", and "$\oplus$".

3 Possibility to acquire new knowledge

As described in the previous section, two types of relations are given to a knowledge base. They are:

- subclass/superclass relation (world3 & world5),
- exclusiveness (world10).

In any world once determined by these relations (i.e. excluding world1), no further information can be attached. In other words, those worlds can not transit to other valid worlds by increasing the number of "f" in their lines).

Considering relations among three *classes*, however, some valid worlds transit to other ones by giving more information. In this case, we have 256 worlds which differ in type of possible *instance* and those are listed in Table 3. For the same reason as Table 2 and for saving a space, we excluded the following worlds from the table.

type of *instance* x	$A(x)$ $\wedge$ $B(x)$ $\wedge$ $C(x)$	$A(x)$ $\wedge$ $B(x)$ $\wedge$ $\neg C(x)$	$A(x)$ $\wedge$ $\neg B(x)$ $\wedge$ $C(x)$	$A(x)$ $\wedge$ $\neg B(x)$ $\wedge$ $\neg C(x)$	$\neg A(x)$ $\wedge$ $B(x)$ $\wedge$ $C(x)$	$\neg A(x)$ $\wedge$ $B(x)$ $\wedge$ $\neg C(x)$	$\neg A(x)$ $\wedge$ $\neg B(x)$ $\wedge$ $C(x)$	$\neg A(x)$ $\wedge$ $\neg B(x)$ $\wedge$ $\neg C(x)$	information about *instance*
world1	t	t	t	t	t	t	t	t	
world12	f	f	t	f	t	t	t	t	$(A \subset C) \wedge (A \oplus B)$
world15	t	f	f	f	t	t	t	t	$(A \subset B) \wedge (A \subset C)$
world20	f	f	t	t	f	t	t	t	$(A \oplus B) \wedge (B \oplus C)$
world24	f	f	f	t	f	t	t	t	$(A \oplus B) \wedge (B \oplus C) \wedge (C \oplus A)$
world28	f	f	t	f	f	t	t	t	$(A \subset C) \wedge (B \oplus C)$ $[\wedge(A \oplus B)]$
world43	t	f	t	f	t	f	t	t	$(A \subset C) \wedge (B \subset C)$
world44	f	f	t	f	t	f	t	t	$(A \subset C) \wedge (B \subset C) \wedge (A \oplus B)$
world47	t	f	f	f	t	f	t	t	$(A \subset B) \wedge (B \subset C)$ $[\wedge(A \subset C)]$

Table 3: Possible world considering 3 *classes*

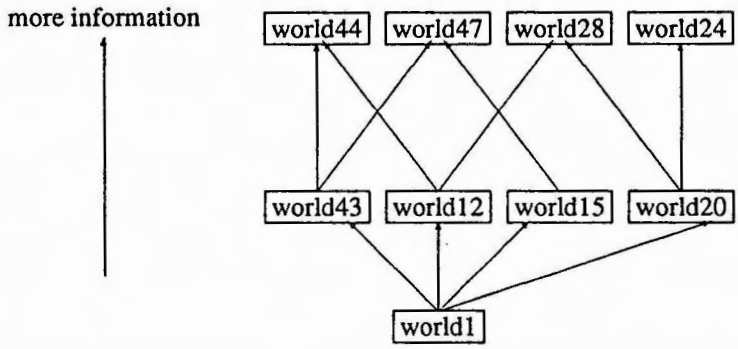


Figure 3: Amount of information given by relations among 3 *classes*

- Worlds which are meaningless in a realistic knowledge base like the case in Table 1.
- Worlds which are impossible to be determined by a combination of binary relations. In "world223", for example, every *instance* necessarily belongs to two *classes* out of A, B, and C. Since, no binary relation can represent a world like that, we excluded also such valid but not representable worlds.
- If several worlds are symmetrical with each other as for *classes* (e.g. world3 and world5 in Table 2), only one of them is listed.

The worlds are ordered as shown in Figure 3 according to the amount of information in each world. The figure is different from Figure 2 in the fact that "world12, 15, 20, and 43" possibly transit to another valid world along each arrow in the figure if some more information is added. By contrast, "world24, 28, 44, 47" have no room of additional information.

$$
\begin{array}{ll}
\text{world43} + (A \oplus B) \Rightarrow \text{world44} & (\textit{exclusion}) \\
\text{world43} + (A \subset B) \Rightarrow \text{world47} & (\textit{specialization}) \\
\text{world12} + (A \oplus B) \Rightarrow \text{world44} & (\textit{exclusion}) \\
\text{world12} + (A \subset B) \Rightarrow \text{world28} & (\textit{categorization}) \\
\text{world15} + (B \subset C) \Rightarrow \text{world47} & (\textit{generalization}) \\
\text{world20} + (A \subset C) \Rightarrow \text{world28} & (\textit{diplomacy}) \\
\text{world20} + (A \oplus C) \Rightarrow \text{world24} & (\textit{exclusion})
\end{array}
$$

Figure 4: Types of knowledge acquisition

What is the meaning of these transitions? Let's consider the following situation:

We know:
A robin is a bird. ("robin" $\subset$ "bird")
A robin flies. ("robin" $\subset$ "flier")
Then the new knowledge is acquired:
A bird flies. ("bird" $\subset$ "flier")

This corresponds to the transition from "world43" to "world47". Since the knowledge that a robin flies is generalized into the knowledge that a bird flies, we call it *generalization*. It can be said that this transition is one of mechanisms of knowledge acquisition.

We show in Figure 4 all types of transitions. Five different knowledge acquisition mechanisms are listed in the figure: they are *generalization, specialization, exclusion, categorization*, and *diplomacy*. We are familiar with the first two mechanism in many methods of concept learning. On the other hand, the rests are not paid attention to so far but as useful and effective as the former. We will describe below how these mechanisms work.

Exclusion is the process to find that two *classes* are exclusive of each other (i.e. have no common instance). There are three variations of *exclusion* in Figure 4. The next is the example of the topmost case of the figure:

We know:
A dog is an animal. ("dog" $\subset$ "animal")
A cat is also an animal. ("cat" $\subset$ "animal")
Then the new knowledge is acquired:
A dog is not a cat. (i.e. they have no common instance ("dog" $\oplus$ "cat")

Categorization is the process in which two exclusive *classes* are found to be different categories of a certain *class*. The following example clarify this process:

We know:
A human is not an ape. ("human" $\oplus$ "ape")
A human is a primate. ("human" $\subset$ "primate")
Then the new knowledge is acquired:
An ape is also a primate. ("ape" $\subset$ "primate")

Diplomacy is the process in which two *classes* that have common exclusive *class* are found to have subclass/superclass relationship. The example situation is:

We know:
A bird is not a mammal. ("bird" $\oplus$ "mammal")

A penguin is not a mammal. ("penguin" $\oplus$ "mammal")

Then the new knowledge is acquired:

A penguin is a bird. ("penguin" $\subset$ "bird")

We have named this mechanism as *diplomacy* because it reminds us a basic strategy in diplomacy: that is *"enemy of enemy is friend"*.

Our system search the knowledge base for the possibility to apply these mechanisms. In other words, these mechanism give the system the motivation to acquire some specific knowledge actively.

4 Knowledge acquisition process

Now we describe how our knowledge acquisition proceeds. The basic idea is that the system searches out the incompleteness in new knowledge, finds out what kind of knowledge is necessary to be added, and then asks and obtains it from human.

In the previous section, we discussed about a mechanism of knowledge acquisition which can occur at an incomplete part of knowledge. It is, however, not realistic to get all insufficient information from human, because new knowledge may have much incompleteness and it can be a long time consuming for human to respond to all questions the system asks.

We proposed a method for evaluating importance of concepts over a semantic network in [Handa 88]. Taking the importance of concepts into account, we have attained effective knowledge acquisition through a simple strategy of asking to a human only about relations between important concepts.

The input data are obtained from newspaper articles. They are at first manually translated into IXL, a semantic network based language for knowledge representation ([Handa 86]), and then given to our system one article by one.

The system maintains a data base called *potential knowledge*) along with the knowledge base. *Potential knowledge* is a set of *potential relations* inferred by the system from each triplet of concepts that are related to each other in the same form as concepts A, B, and C in Table 3. The system tries to prove them to be true or false both with the new knowledge and through a question to a human.

The process of knowledge acquisition is triggered when new knowledge obtained from one article are given to the knowledge base. The steps of the process are as follows:

[Consistency check]

1. Check the inconsistency of the new knowledge with the already existing knowledge.
2. Try to prove relations of a potential relation list with this new knowledge. If some of them are proved to be true, append them to the new knowledge and reduce redundancy in the modified new knowledge.

[Select important questions]

3. Calculate importance values of concepts in the new knowledge.

4. Search for possibility of appending new relations both in the new knowledge and at the border of the new and existing knowledge. We call this new relation as a "potential relation".

5. Calculate the importance value of each potential relation by appending importance values of concepts related to the relation, then sort them according to the values.

[Getting knowledge from a human]

6. Ask a human if each potential relation is true or not starting from the most important one until the human rejects answering.

7. Assimilate the information obtained from the human into the new knowledge. If some potential relations turn out to be false by this assimilation, exclude them from the list.

8. Keep the list of unanswered ones of potential relations for the later chance.

The dialogue of Figure 5 explains how this process proceeds. As of the first step of the figure, the system has already acquired knowledge about several workstations (e.g. Sun's and DEC's machines) and has known that such concepts as EWS, CPU, and OS are important. And the new knowledge is as for a new workstation of YHP (Yokogawa Hewlett Packard). Sentences following a prompt "==> are typed by the human. Italic sentences are comments.

At *Step 1* in the figure, since the system found that new knowledge "MC68020 is CPU" caused redundancy, the redundant relation was deleted from the knowledge base.

At *Step 2*, the new knowledge also proved that some *potential relations* were true and another ones false. The system maintained potential knowledge accordingly.

At *Step 6*, the system tried *exclusion* both of 'YHPmachine' and of 'MC68020'. The first succeeded but the next failed. Then it tried *specialization* of 'MC68020' and succeeded. The last is *generalization* about 'products' of 'YHP' and failed.

One of the advantages of this system is that we don't have to organize new knowledge carefully so that it contains just enough information for knowledge assimilation. This is important when we build knowledge base from textual documents such as newspaper articles. Although these sources generally assume much background knowledge, it's difficult for a human to add all of them to input data. With our method, the system automatically do a certain extent of this job.

Another point is that the system requires only important information, which considerably reduces the human lord. Although there may remain many unresolved questions in the resulting knowledge base, we still have a chance to solve them by giving another knowledge to the system in the future.

We processed 70 articles with this system. On each article, we allowed the system to ask 5 questions. The resulting knowledge base consisted of 7420 nodes and 12045 links (2117 links are created from the questions). Table 4 shows how many times each mechanism worked.

```
Reading new knowledge...done.
150 nodes 214 links created.
Inconsistency and redundancy check...                    Step 1
Old knowledge:
     isa('CPU', micro_chip), isa('MC68020', micro_chip)
New knowledge:
     isa('MC68020', 'CPU')
Deleted knowledge:
     isa('MC68020', micro_chip)
Maintaining potential knowledge...                       Step 2
Deleted potential knowledge:
     isa('MC68020', 'CPU') -- proved
     exclusive('CPU', 'MC68020') -- inconsistent
     isa('MC68020', 'CPU') -- inconsistent
      ...
Calculating importance values of concepts...done.       Step 3
Making potential relation list....done.                  Step 4 & 5
Questions about potential relations...                   Step 6 & 7
isa(_220, 'EWS'), exclusive('YHPmachine', _220) ?        Is 'YHPmachine' exclusive
                                                         with some other subclasses
                                                         of 'EWS'?
     ==> yes.
New knowledge:
     exclusive('YHPmachine', 'SUNmachine')
     exclusive('YHPmachine', 'DECmachine')
      ...
Deleted potential relation:
     isa('YHPmachine', 'SUNmachine')
     isa('YHPmachine', 'DECmachine')
      ...
isa(_224, 'CPU'), exclusive('MC68020', _224) ?
     ==> show(X,isa(X, 'CPU')).                          Show subclasses of 'CPU'.
     'Intel_family', 'Motorolla_family', 'SPARC', 'VM2020', '6809'
     ==> no.
isa(_270, 'CPU'), isa('MC68020', _270) ?                 Is 'MC68020' a subclass
                                                         of another subclass of 'CPU'?
     ==> 'Motorolla_family'.
New knowledge:
     isa('MC68020', 'Motorolla_family')
Deleted knowledge:
     isa('MC68020', 'CPU')
relation(produce, 'YHP':self, 'EWS':all) ?
     ==> no.
exclusive('UNIX', 'HP-UX') ?
     ==> end.
28 potential relations are still in the list.            Step 8
```

Figure 5: Dialogue with the knowledge acquisition process

mechanism	total	success	fail
exclusion	187	132	55
specialization	89	50	39
generalization	50	16	34
categorization	18	9	9
diplomacy	6	4	2

Table 4: Frequency of each knowledge acquisition mechanism

Frequency of *exclusion* and its high percentage of success are due to the tendency of the source articles. There are numbers of "... is a ..." type descriptions compared with "... is different from ..." type descriptions. This leads to many superclass/subclass relations in the knowledge base, and hence leads to many chances of finding "world15" and "world43" shown in Table 3. Although Figure 4 shows the possibility of *exclusion, specialization* and *generalization* at these worlds, the *exclusion* occurs most frequently because concepts that appear in the newspaper articles tend to be classified into different categories strictly.

This experiment made it clear that newspaper articles had much ambiguity as for classification of concepts but our system could create a well-formed knowledge base by resolving the ambiguity with support of a human instructor.

5 Conclusion

We proposed in this paper "acquisition of relation between concepts" as one form of concept learning. At first, we provided a logical basis for discussing the kind of information involved in a relation between concepts. Next, knowledge acquisition mechanisms are classified into 5 different types according to a type of information required for them. Finally, we showed how these knowledge acquisition took place in the course of building the knowledge base. Considering importance of relations required for knowledge acquisition resulted in an effective and realistic process.

There still remain several problems to be solved in the future. These are:

- We have assumed that all relations in a knowledge base are binary (i.e. relation between two concepts). As the result, we could exclude several worlds from the consideration. In a real world, however, we inevitably face more complicated knowledge which are not represented by any combinations of binary relations. We should construct a theory that meets with such information.
- The system uses only deductive reasoning in the current process of knowledge acquisition. Introducing inductive reasoning using information of *instances* will improve the performance of the system.
- There is yet possibility to improve the way of dialogues with a human. "Yes or no" question is surely easy to answer but sometimes irritating. We should develop a technique to obtain much more information from the human by fewer questions.

The method of learning relations between concepts helps us not only in the field of building

large scale knowledge base, but in general fields of knowledge processing. We would like to apply this method also to such fields as expert systems, problem solving, and natural language processing.

References

[Handa 86] Handa, K., Higuchi, T, et al., "Flexible Semantic Networks for Knowledge Representation," Journal of Information Processing, Vol.10 No.1, 1986.

[Handa 88] Handa, K. and Ishizaki, S., "Learning Importance of Concepts: Construction of Representative Network," Proc. of 3rd Knowledge Acquistion for Knowledge-based Systems Work shop, 1988.

[Lebowitz 86] Lebowitz, M., "Concept Learning in a Rich Input Domain: Generalization-Based Memory," in "Machine Learning Volume II," Michalski, R.S., et.al. (Ed.), Morgan Kaufmann, California, 1986.

[Michalski 80] Michalski, R.S., "Pattern Recognition as Rule-guided Indective Inference," IEEE Trans. on PAMI, Vol.2, No.4, pp.349-61, 1980.

[Winston 72] Winston, P.H., "Learning Structual Descriptions from Examples," in "The Psychology of Computer Vision," Winston, P.H. (Ed.), McGraw-Hill, New York, 1972.

Malefiz - A Learning Apprentice System that Acquires Geometrical Knowledge about a Complex Design Task

Jürgen Herrmann
Renate Beckmann

Universität Dortmund
FB Informatik I
Postfach 50 05 00
4600 Dortmund 50
F. R. Germany

Abstract

In this paper the machine learning system Malefiz is presented. It acquires knowledge about floorplanning, a task that belongs to the early phases of VLSI design. In the first part of this paper the principles of floorplanning are described briefly. The characteristics of knowledge about floorplanning are explained. Then the architecture of Malefiz is described. Its working method as a learning apprentice system is pointed out. With an example the different types of knowledge used by Malefiz are explained.

1. Introduction

In recent years the interest in knowledge acquisition for expert systems has grown significantly. Several tools have been developed that support knowledge acquisition for different classes of problems. For different reasons only little research has been concerned with the acquisition of knowledge for complex constructive problems. In the classification of knowledge acquisition tools in [1] no tools are mentioned that belong to the class *design*.

For machine learning programs that acquire knowledge for expert systems the situation is similar. The few systems that deal with design problems are restricted to simplified tasks that have the complexity of toy problems or work on small subproblems of design tasks that have a well understood and complete domain theory [2,3,4]. Concepts for the application of machine learning techniques to a less restricted problem (the generation of architecture plans for VLSI chips) are described in [5]. This paper deals with another complex design problem: Floorplanning for integrated circuits.

2. Floorplanning

The Task

Floorplanning is one of the early phases of VLSI design. During this phase a geometrical hardware description is synthesized from a structural one (net list of functional blocks on register-transfer level). In [6] floorplanning is defined in the following way:

"In rough estimation, areas are allotted to blocks which have been defined in accordance with functional requirements; those areas are then connected via the wiring areas, keeping in mind data and control flows; thus the overall chip design (*floorplan*) is created."

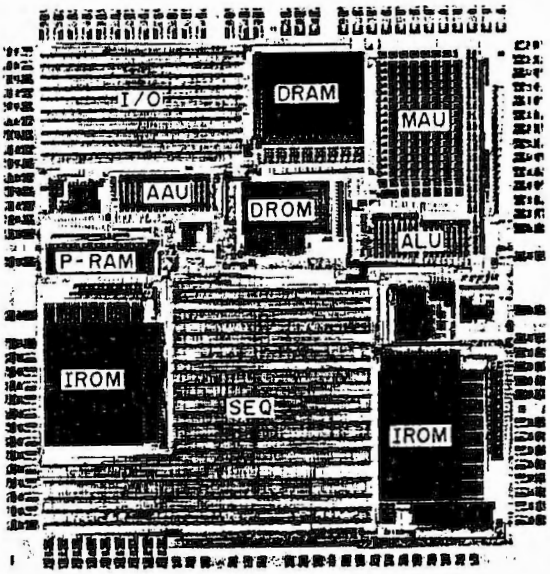


Figure 1: Example of a floorplan [7]

Usually a floorplanning task is formulated as a structural hardware description. In this paper for each of the functional blocks the following data are considered:

- Description of the function
- Extensions of the block
- Connections to other blocks (for each connection a number is delivered that describes its significance).

Floorplanning is guided by the following goals:

- Minimize the overall chip area
- Place blocks with significant connections close together.

Characteristics of Knowledge about Floorplanning

Before the Malefiz system is descibed a closer look at the floorplanning task is necessary. The architecture of Malefiz is shaped by the characteristics listed below.

- *Floorplanning can be described adequately by state-space search*
  In contrary to many other synthesis tasks (e.g. chip-architecture planning [5]) the initial and the goal states of a floorplanning process can be described as elements of a *single* state-space. The state-space for floorplanning can be organized in the following way: An initial state consists of an empty planning area and a list of unplaced blocks that contains all blocks. A goal state consists of a planning area on which all blocks are placed and an empty list of unplaced blocks. For the intermediate states some blocks are placed on the planning area and some are left in the list of unplaced blocks. There is a set of operators. Each one can be applied to certain states and proceeds the floorplanning process in this way. Floorplanning operatord are e.g.:
  - Select the next block that shall be placed
  - Place block X at the left side of block Y
  - Move block Y left
- *Floorplanning is an optimization problem*
  The different solutions for a floorplanning problem need not to be equivalent. They can have different *qualities*. The quality of a solution (a goal state) depends on the goals of the floorplanning process mentioned above. This means that the solution path in the state-space should not lead to "any goal state" but to one with an acceptable quality. This is an important difference to most other machine learning systems that base on state-space search (e.g. [8,9]). (The goal states of a floorplanning task with an unacceptable quality can be considered as intermediate states that have to be optimized by the application of modification operators like "Move block Y left".) The acceptability of a goal state (and also of an intermediate state) can be checked by an evaluation function that calulates a numerical value for each state.
- *Geometrical knowledge is needed for floorplanning*
  A floorplanning problem is solved mainly geometrically. To keep the used planning area small the blocks have to be placed as close together as possible, i.e. the unused area between the blocks must be minimized. Geometrical knowledge, i.e. knowledge about the analysis and manipulation of geometrical structures, is necessary to achieve this goal.
  The representation and acquisition of geometrical knowledge has already been addressed by other machine learning systems before. An example is the well-known blocks-world manipulation [10,11]. The difficulties with the representation of geometrical floorplanning knowledge are caused by the following reasons:
  a) There are multiple different extensions for the rectangular blocks
  b) Not only the topological relations between objects (neighbourhood etc.) but also the exact geometrical coordinates determine the floorplanning operations.
  c) The geometrical concepts the reasoning of the expert bases on must be represented in full generality. The are e.g. various different kinds of gaps between blocks that are all special cases of the concept "gap".
  d) It is not feasible to represent all relevant information explicitly in each state description. This

would lead to a combinatorical explosion of the length of the description. This means that much information about a floorplanning state must be extracted. Data like the relation of used and unused area must be derived from a state description because they are represented implicitly.

- *There is no domain theory about floorplanning*
  Usually floorplanning is performed by IC design experts that have acquired their knowledge from practical experience. Their way of problem solving is intuitive and empirical. Because of that deductive knowledge about floorplanning is lacking.

3. Related Work on Learning Apprentice Systems

Learning apprentice systems are defined in [2] as "... the class of *interactive* knowledge-based consultants that directly assimilate new knowledge by observing and analyzing the problem solving steps contributed by their users through their *normal* use of the system." Several systems have been developed that belong to this class of learning systems. LEAP [2] acquires rules about the transformation of a functional hardware description into a gate level description. The generalization of an example given by the user is based on a verification of the correctness of this example. This method is not applicable to floorplanning because it cannot be verified that the application of a floorplanning operator is optimal or "suitable". The knowledge that would be needed for this verification is lacking. For similar reasons Wilkins apprenticeship learning method [12] and the analytical learning method DISCIPLE [13] is based on cannot be applied to floorplanning. The main observable of Wilkins' ODYSSEUS system is to find out feature values that are used for heuristic classification. The application of a floorplanning operator with several parameters is a much more complex task that cannot be modelled in this way appropriately. DISCIPLE uses semantic networks to explain an example. This kind of declarative background knowledge is lacking for floorplanning.

4. Architecture of a Learning Apprentice System for Floorplanning

Malefiz learns rules about the construction of a floorplan. It is organized as a learning apprentice system.

Because of the lacking domain theory about floorplanning Malefiz does not use analytical learning methods. Instead similarity-based learning methods are applied. Figure 2 shows the structure of the Malefiz system.

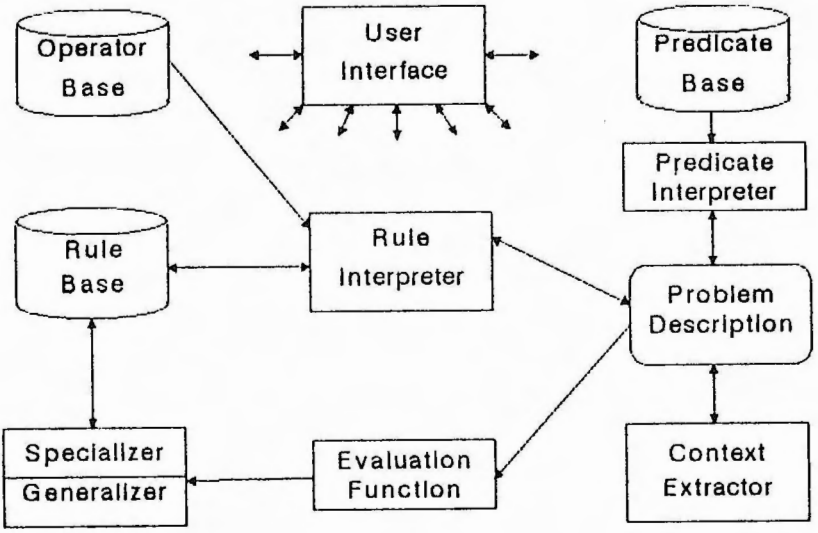


Figure 2: Structure of the Malefiz system

The Components of the Malefiz System

In the *problem description* the current state of the problem solving process is represented. The already placed blocks as well as the list of unplaced blocks are described. This information is represented in a structured form. A LISP-based flavor system [14] is used for this purpose that describes besides other things the topological relations (neighbourhood, relative location of two blocks) of the placed blocks. The geometrical information about the floorplanning state that needs large computational effort for its analysis and manipulation is represented with Corner-Stiching data structures [15] that are implemented in C. The LISP-based and the C-based parts of the representation are connected by pointers. After the application of the extractor and the predicate interpreter (see below) the problem description contains also a logic-based description (a list of predicates) of a part of the current state. This description is suitable for the manipulations performed by the generalizer and the specializer (see below).

The *user interface* supports the interaction of the user and the system. Using it the designer can observe all components of Malefiz and answer questions that are generated by the system. Besides that the user interface supports the modification of the acquired and the given knowledge (operator base, predicate base).

The *operator base* contains all known operators that are needed for the manipulation of a floorplanning state.

The *context extractor* cuts out that part of the floorplanning state that is relevant for the next application of an operator. For each operator application only a part of the state is considered, because otherwise the logic-based description that is generated from the state description would become too big. This would hinder or prevent the application of the generalizer and the specializer.

In the *predicate base* the predicates are listed that are needed for the generation of the logic-based description. By means of the predicates information becomes explicit that is only represented

implicitly in the C- and the LISP-based description.

The *predicate interpreter* applies the predicates to the part of the state description that is delivered by the context extractor and stores the result in the problem description.

The *rule base* contains the rules that are currently known. New rules that are learned by Malefiz are stored here.

The *rule interpreter* has two functions: It determines the rules that are currently applicable and selects the most promising one according to the rule selection strategy. The operator that belongs to this rule or another one that is selected by the user is applied. This operation leads to a new floorplanning state.

The *generalizer* operates on the rule base. It looks for pairs of rules that have identical right hand sides (operator descriptions) and generalizes them by use of the *dropping condition* and the *climbing generalization tree* generalization operators [16]. If a generalization is successful the generalized rule substitutes the more specific ones.

The *evaluation function* puts values on the results of rules that have been generated by the generalizer. If the state that results from the application of a specific rule is not acceptable according to the goals of the floorplanning process the rule must have been over-generalized before.

The *specializer* is applied to an over-generalized rule. It specializes it by excluding the erroneous state explicitly. The state description is inserted into the WITHOUT part of the rule. The WITHOUT part describes the states the rule is not applicable to. The WITHOUT part is not generalized automatically by the system because over-generalization could take place. (IF the user decides that a generalization of the WITHOUT part is nesessary he/she can apply the generalizer to it.)

The Working Method of the Malefiz System

Malefiz assists the user in the following way: At the beginning of a session the user puts the description of a floorplanning problem into the system. The system translates these data into the structural and geometrical representations that base on flavors and Corner-Stitching. The user needs not bother with this translation step. After that the problem description contains the initial state for the floorplanning process. Now the following cycle is performed repeatedly:

First the kind of operation that shall be performed next is destined by the user. Malefiz supports three kinds of operations: Placing a block, moving a block and removing a previously placed block.

As a next step the context for the following (proposed) rule application is destined by the context extractor. For the initial state that has an empty planning area the context consists only of the list of unplaced blocks. For an intermediate state the context is derived from the neighbourhood of the least recently placed block or of the block with the most significant connection to an unplaced block.

Now the predicate interpreter creates the logic-based representation of the context. It uses the predicate base for this purpose. Using this logic-based description the rule interpreter destines the rule that can be applied to it.

If there are applicable rules the rule interpreter selects the most promising one according to its (built in) strategy and proposes the operator that belongs to the right-hand side of the rule as the next floorplanning step.

The context extration, the creation of the logic based description and the rule selection are hidden

from the user. He/she is only confronted with the result of these actions: the proposed operator.

Now the user has the choice. He/she can select the proposed operator or reject it. If he/she accepts the operator it is applied to the structural and the geometrical representation of the floorplan. In this way a new state of the problem solving process is reached. It is checked by the evaluation function. The user interface shows the new state of the planning area to the user in a graphical form.

If the new state is characterized as "not acceptable" by the evaluation function an the user confirms this it can be concluded that the applied rule was unsuitable for the old state. It must have been over-generalized in one of the previous generalization steps. Because of that the rule application is reset and the specializer is called that makes the left-hand side of the rule more specific to exclude the old state. This is performed by insertion of the old state description into the WITHOUT part of the rule. These action are happening in the background of the system. The user needs not control it. The user interface shows him/her that the operation has been reset.

If no applicable rule is available or if the user rejects the proposed one Malefiz asks him/her to propose an alternative operator application (that may refer to the same context or a different part of the planning area). The operator application is performed and illustrated by the user interface.

The operator application initiated by the user is evaluated by Malefiz. This evaluation and the following learning step are also hidden from the user. A logic-based context description is formed for the part of the planning area that has been modified by the applied operator. If there is no rule that has the corresponding operator as a right-hand side a new rule is formed that receives the context description as a left-hand side. This rule is inserted into the rule base. Otherwise the left-hand side of the existing rule (there is at most one rule for each operator) and the new context description are passed to the generalizer. Using the dropping condition and the climbing concept hierarchy tree generalization rules it creates a generalization of the two descriptions. (The example descriptions contain much more predicates than are needed for the generalized rule. Because of that in contrary to methods like structural matching [17] the dropping condition generalization rule must be used heavily.) In this way a new rule is formed that receives the applied operator as a right-hand side and substitutes the old rule.

From the designers' point of view Malefiz is a design-assistant. (He/she needs not bother with the learning operations that Malefiz is performing.) Malefiz offers a framework for proceeding a floorplanning task. It can represent and display the current floorplanning state. It offers operators the user can apply to perform the floorplanning process in this way. In this sense Malefiz works as a comfortable floorplanning "editor". In addition Malefiz makes proposals for the next operation by application of rules that have been acquired previously. This is the typical behaviour of an assistant that can perform the instructions given by the expert and can make single decisions like the selection of the next rule independantly (controlled by the expert). The learning capabilities make Malefiz an apprentice that acquires new knowledge by observing and assisting the proceeding of specific floorplanning problems. The user needs not bother with the learning mechanism. He/She can concentrate on the floorplanning tasks.

5. Example for the performance of Malefiz

Twenty blocks shall be placed on a square area with edges that are 200 units long. The input of the task is given as a list of three lists:

The first list describes the planning area by giving its name, length and width. In the next list the names and extensions of the blocks are given. The last list describes the communications between blocks. For each pair of communicating blocks a numerical value describes the significance of the connection.

This input is converted into a combined frame-based and Corner-Stitching description. Figure 3 shows the frame-based representation of a block that has already been placed.
<block f1>, an object of flavor block has instance variable values:

name:	a1	{name of the block}
X-ext:	40	{width of the block}
Y-ext:	20	{length of the block}
X-pos:	50	{X-position of the lower left corner}
Y-pos:	0	{Y-position of the lower left corner}
conn:	(("a2" 5) ("a4" 2))	{connections to other blocks}
sum-conn:	7	{sum of the values for the connections}
sum-placed-conn:	7	{sum of the connections to placed blocks}
cost-pointer:	959748	{reference to the C data structures}

Figure 3: Frame-based representation of a placed block

Now an intermediate state of the floorplanning process is explained. 16 blocks have been placed already. Figure 4 shows the graphical representation of the state:

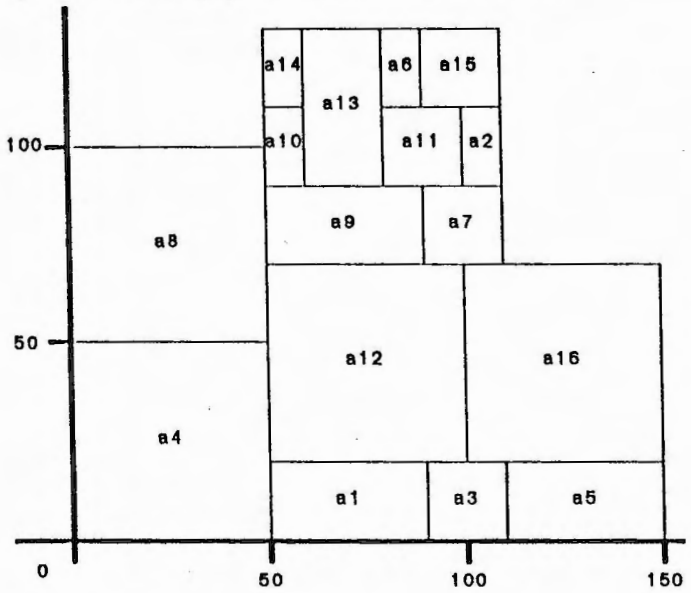


Figure 4: Floorplan with 16 placed blocks

The user destines that the next kind of operation shall be the placement of a block. The block to be

placed next shall be a19. The placed block that has the most significant connection to a19 is a7. Because of that the surrounding of a7 is taken as the context for the operation. The extension of the context is derived from the extensions of a19 and a7 (maximum of the widths and lengths). All blocks that have a maximum distance of 20 from a7 are selected (see Figure 5).

As a next step the logic-based description of the context is formed. The predicates have the following meaning:

"nearly-ready" denotes that more than half of the blocks have already been placed. A floorplan is big ("big-fp") if the product of its extensions is more than the double of the sum of the products of the extentions of the blocks to place on this floorplan. A block B1 is placed "from-the-right" B2 if the X-coordinate of the lower left corner of B1 is greater than the X-coordinate of the lower right corner of B2. "over" is defined accordingly. Two blocks are "side-by-side" if they have a part of an edge in common. A block is "long" if the aspect ratio is bigger than or equal 2. Otherwise it is "square-like". A block is placed "standing" if its extension in Y-direction is bigger than that of the X-direction. Otherwise it is placed "lying". Predicates like "big" and "many-connections" are destined by a comparison of the value for the considered block with the average value. The predicates "many-pconnections" and "max-pconnection" deal with the connections of a block to placed blocks. (The meaning of the other predicates can be derived directly from their terms.)

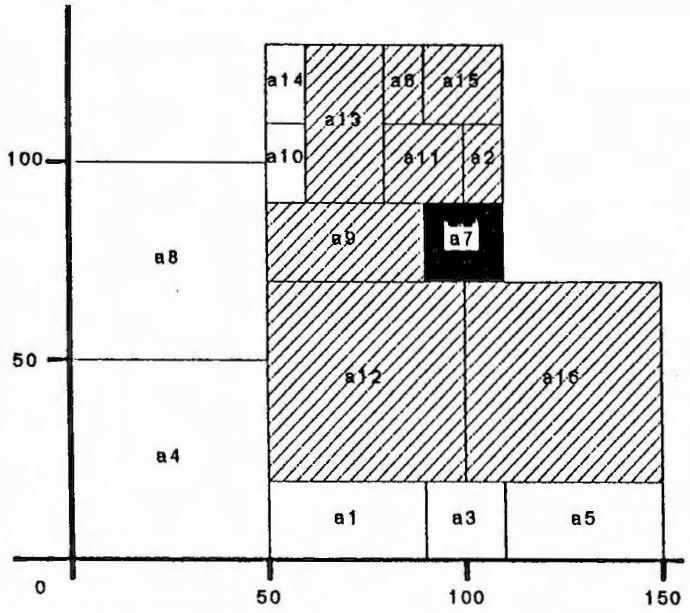


Figure 5: Context for the next placement operation

For the context shown in Figure 5 logic-based description is destined. A part of this description is shown here:

((nearly-ready "fpa") (little_fp "fpa")
(over "a7" "a16") (side-by-side "a7" "a16") (over "a2" "a7") (side-by-side "a2" "a7")
(over "a2" "a16")
(placed "a7") (small "a7") (square-like "a7") (many-connections "a7") (lying "a7")

(unplaced "a19") (small "a19") (square-like "a19") (many-connections "a19")
(many-pconnections "a19") (max-pconnections "a19")
(placed "a16") (big "a16") (square-like "a16") (many-connections "a16") (lying "a16")
(strong-connections "a7" "a19")
(fits-from-the-right-lying "a19" "a2") (fits-from-the-right-standing "a19" "a2")
(fits-from-the-right-lying "a19" "a7") (fits-from-the-right-standing "a19" "a7")
(fits-from-the-right-lying "a19" "a16") (fits-from-the-right-standing "a19" "a16")
(fits-over-lying "a19" "a16") (fits-over-standing "a19" "a16")
(max-connections "a7" "a19")
...)

In our example there is no rule available for this context. Because of that the user has to deliver an operator for the block placement:

(place-lying-from-the-right "fpa" "a19" "a7")

As the term for the operator implies it places the block "a19" to the right of the block "a7" on the floorplan "fpa". The application of this operator leads to the new floorplanning state that is depicted in Figure 6.

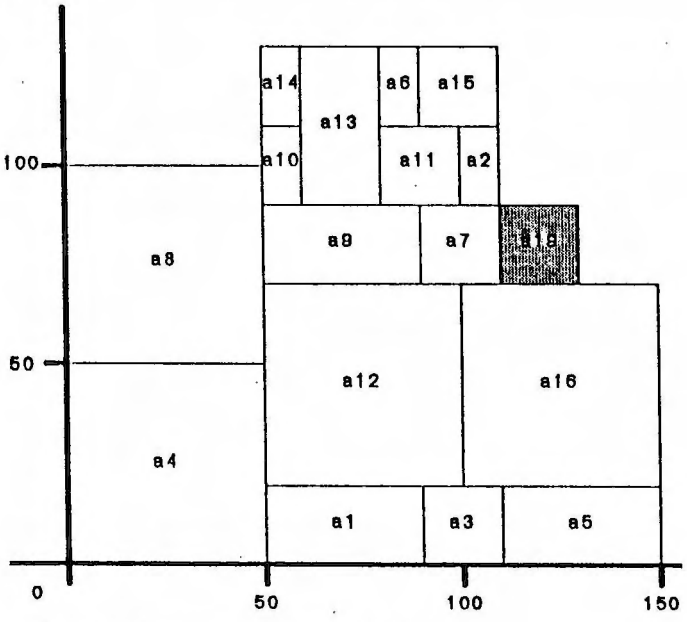


Figure 6: Floorplan after the placement of block a19

Malefiz creates a new very specific rule that gets the context description including the description of the block to place next (here block "a19") as left-hand side and the applied operator as right-hand side.

Now the generalizer is activated. It looks for another rule (in the rule base) that has the same operator as right-hand side. In our example it finds the following one:

IF ((*nearly-empty* "fpb") *(big-fp "fpb")*
(over "b7" "b16") (side-by-side "b7" "b16")
(over "b2" "b7") (side-by-side "b2" "b7")
(placed "b7") (small "b7")
(long "b7") (many-connections "b7") (lying "b7")
(unplaced "b19") *(big "b19")*
(square-like "b19") (many-connections "b19")
(many-pconnections "b19") (max-pconnections "b19")
(placed "b16") (big "b16")
(square-like "b16") (many-connections "b16") (lying "b16")
(strong-connections "b7" "b19")
(fits-from-the-right-lying "b19" "b7")
(fits-from-the-left-standing "b19" "b7")
(max-connections "b7" "b19")
)
WITHOUT ()
THEN OP-i: (place-lying-from-the-right "fpb" "b19" "b7")

These two rules are generalized by application of the dropping condition and the climbing generalization tree operators. In the last rule all parts are printed in italics which are generalized by one of the two operators. This leads to the following rule that is less specific than the two other ones:

IF ((just-started "fpb")
(over "b7" "b16") (side-by-side "b7" "b16")
(over "b2" "b7") (side-by-side "b2" "b7")
(placed "b7") (small "b7")
(many-connections "b7") (lying "b7")
(unplaced "b19")
(square-like "b19") (many-connections "b19")
(many-pconnections "b19") (max-pconnections "b19")
(placed "b16") (big "b16")
(square-like "b16") (many-connections "b16") (lying "b16")
(strong-connections "b7" "b19")
(fits-from-the-right-lying "b19" "b7")
(max-connections "b7" "b19")
)
WITHOUT ()
THEN OP-i: (place-lying-from-the-right "fpb" "b19" "b7")

This rule substitutes the two other ones that have the same right-hand sides. In this way the learning step is finished and a new cycle can begin.

After creating a lot of little floorplans Malefiz has learned rules like:

```
IF ((just-started "fpx") (big-fp "fpx")
    (placed "x7")
    (unplaced "x19")
    (max-pconnections "x19")
    (strong-connections "x7" "x19")
    (fits-from-the-right-lying "x19" "x7")
    (max-connections "x7" "x19")
  )
WITHOUT (...)
THEN OP-i: (place-lying-from-the-right "fpb" "b19" "b7")
```

or

```
IF ((empty "fpx")
    (unplaced "x19") (many-connections "x19")
  )
WITHOUT (...)
THEN OP-i: (place-in-the-middle "fpx" "b19")
```

6. Conclusions

A prototype of Malefiz has been implemented at the University of Dortmund, Lehrstuhl Informatik I (Prof. Dr. B. Reusch). In this way it has been shown that knowledge about a complex, highly constrained design task can be acquired with machine learning methods. In spite of the current restrictions for Malefiz the system shows the applicability of these methods to kinds of tasks that are not or only scarcely considered by the current machine learning research. (The Franz Lisp interpreter limits e.g. the number of predicates that can be manipulated by the generalizer. Because of that only small example floorplans could be used.)

Malefiz is organized as a learning apprentice system. Because of that it can be used directly by the floorplanning expert. For him/her the system operates as a design assistant that can perform the floorplanning operators, display the floorplanning state and make proposals for single floorplanning steps. The learning mechanism does not depend on user interaction.

During the development of Malefiz one main difficulty has been the representation of geometrical and structural knowledge. It became clear that a logic-based description is not sufficient for this purpose. On the other hand the logic-based description is needed for the similarity-based generalization methods used by Malefiz. Because of that a hybrid representation has been selected. The manipulation of the geometrical data is performed by the (predefined) floorplanning operators. It would not be feasible to learn and represent the details of the floorplanning operations based on a floorplanning description with predicates.

The separation of the structural (flavors) and the geometrical information (Corner-Stitching) is not necessary. Both kinds of information can be repesented adequately in a LISP-based description. For Malefiz the separation was selected because in this way existing software could be integrated into the

system (Corner-Stitching implementation in C).

A more comfortable and probably more capable solution would be the use of generalization methods that work directly on the structural/geometrical repesentation. The development of appropriate methods for this task is an important goal for further research.

Even the manual acquisition and representation of geometrical knowledge is difficult for expert systems that address tasks similar to floorplanning [18,19]. It is e.g. difficult to formulate rules that are general enough to manipulate a class of similar geometrical situations. Because of that expert systems that address problems of this kind typically are very constrained.

The concept of a context for a rule application enables Malefiz to consider only the relevant parts of the floorplanning states for generalization. This concept is essential for the learning performance of Malefiz. Further research is needed to find out if the context can be destined and used in the same way for floorplanning tasks that depend on additional goals and constraints (timing, power consumption etc).

References

[1] J.H. Booze
A research framework for knowledge acquisition tools
Proceedings of the European Knowledge Acquisition Workshop (EKAW '88)
GMD-Studies Nr. 143

[2] R.S. Mitchell, S. Mahadevan, L. Steinberg
LEAP: A learning apprentice for VLSI design
Proceeding of IJCAI 1985

[3] R.H. Lathrop, R.S. Kirk
A System Which Uses Examples To Learn VLSI Structure Manipulations
National Conference on Artificial Intelligence, AAAI 1987

[4] S.W. Norton, K.M. Kelly
Learning Preference Rules for a VLSI Design Problem Solver
Proc. of Forth Int. Conference on AI Applications, 1988

[5] J. Herrmann, H. Franzke
Requirements for Computer-Aided Knowledge Acquisition in an Ill-Structured Domain: A Case Study
Proceedings of the European Knowledge Acquisition Workshop (EKAW '88)
GMD-Studies Nr. 143

[6] E. Hörbst, C. Müller-Schloer, H. Schwärzel
Design of VLSI Circuits Based on VENUS
Springer 1987

[7] T. Kimura et al.
A Dynamic Time Warping VLSI Processor with a Hierarchical Control Structure
Proc. of the International Conference on Very Large Scale Integration, VLSI 85

[8] B.W. Porter
Learning Problem Solving
PhD-Thesis at the University of California, Irvine 1984

[9] L. Randell
Conceptual Knowledge Acquisition in Search
in: Computational Models of Learning, L. Bolc (ed.)
Springer Verlag 1987

[10] P.H. Winston
Artificial Intelligence, pp 391-414
Addison-Wesley 1984

[11] C. Sammut, R.B. Banerji
Learning Concepts by Asing Questions
in: Machine Learning Volume II, R.S. Michalski, J.G. Carbonell and T.M. Mitchell (eds.)
Morgan Kaufmann Publishers 1986

[12] D.C. Williams
Knowledge Based Refinement Using Apprenticeship Learning Techniques
in: Knowledge Representation and Organization in Machine Learning, K. Morik (ed.)
Springer Verlag 1989

[13] Y. Kodratoff, G. Tecuci
The Central Role of Explanations in DISCIPLE
in: Knowledge Representation and Organization in Machine Learning, K. Morik (ed.)
Springer Verlag 1989

[14] D. Weinreb, D. Moon
Flavors: Message Passing in the Lisp Machine
MIT Artificial Intelligence Laboratory, A.I. Memo No. 602, 1980

[15] J.K. Ousterhout
Corner Stitching: A Data-Structuring Technique for VLSI Layout Tools
IEEE Transactions on Computer-Aided Design of Integrated Circuits, Vol CAD-3, No 1 Jan. 1984

[16] R.S. Michalski
A Theory and Methodology of Inductive Learning
in: Machine Learning, R.S. Michalski, T.M. Mitchell and J.G. Carbonell (eds.)
Tioga Publishing 1983

[17] Y. Kodratoff, J.-G. Ganascia
Improving the Generalization Step in Learning
in: Machine Learning Volume II, R.S. Michalski, J.G. Carbonell and T.M. Mitchell (eds.)
Morgan Kaufmann Publishers 1986

[18] R. Brück, J. Herrmann
KOCOS - An Expert System for VLSI Layout Compaction
Proc. of Intern. Conference on Computer Design, ICCD 1986

[19] M. Kloth, J. Herrmann
EXIST - An Expert System for the Innerplant Planning of Locations
in: Knowledge-Based Expert Systems in Engineering, D. Siriam and R.A.Adey (eds.)
Computational Mechanics Publications, 1987

Evaluating Interaction in Knowledge Acquisition: a Case Study

David L. Maulsby, Gregory A. James and Ian H. Witten

Knowledge Sciences Laboratory, Department of Computer Science
The University of Calgary, 2500 University Drive NW
Calgary, Canada T2N 1N4

phone: (403) 220-6015
email: maulsby@cpsc.UCalgary.CA, ian@cpsc.UCalgary.CA

Abstract

One important feature of a knowledge acquisition system is its method of elicitation. Critical aspects of this should be evaluated in isolation from the system's user interface, and even before the system is implemented. The Metamouse system acquires procedural descriptions of graphical concepts; its elicitation method is apprenticeship learning. The method is explained to users in terms of a metaphor — a short-sighted, touch-sensitive graphical turtle. Key features of the metaphor are the distinctions it makes between touches in order to identify relevant constraints on actions, and the criteria for matching one action to another in order to predict subsequent steps. This paper describes experiments conducted on this method of elicitation. The metaphor is isolated from the system by using questionnaires that depict the turtle's behavior and ask the subject to predict the distinctions it makes or the actions it performs. The paper also presents experiments in teaching concepts to the actual implemented system.

Keywords: knowledge acquisition, apprenticeship learning, metaphor.

1 Introduction

Previous work on evaluating knowledge acquisition systems has concentrated on the validity of the resultant knowledge base [Gruber 87, Shaw 88, Benbasat 88]; less attention has been given to the quality of interaction during elicitation, even though this may profoundly affect the value of the knowledge acquired. Interaction is especially important in systems that build heuristically generalized models. Knowledge acquisition systems are generally targeted at the casual end-user, and interactive elicitation techniques should meet appropriate ease-of-use criteria. Moreover, user and system should have compatible representations of objects, attributes, relations, inference and other elements of knowledge.

This paper describes evaluation studies performed on a procedural knowledge acquisition system based on an apprenticeship metaphor [Maulsby 88a]. Adopting the constructive approach to specifying constraints, as in [Fuller 88] and [Noma 88], it learns procedural descriptions of graphical concepts, such as object structures, spatial relations and transformations. Owing to the system's complexity it is difficult to evaluate *in toto*, as is virtually any knowledge acquisition method. This problem is approached by isolating features of the system judged to be especially important or sensitive and studying their effect upon users.

The system's elicitation device is a metaphorical apprentice which embodies a model of spatial relations and rules to classify graphical constraints according to their importance in governing actions. The metaphor has been studied using questionnaires in which potential users were asked to predict the apprentice's actions and the discriminations it makes between situations it encounters; a pilot experiment was reported in [Maulsby 88b]. This approach yields a measure of compatibility between user and system in terms of the user's understanding of the system's choice of discriminations and

inference criteria. This measure is convolved (inevitably) with the user's ability to learn similar discriminations and inferences. The metaphor's effectiveness can thus be represented on two scales: 1) the rate at which users correctly predict the apprentice's responses (ie. the level of comprehension of the metaphor); and 2) the rate at which users improve in their predictions to some level of comprehension.

Following a review of the system and its elicitation device, the *Metamouse,* we present an account of validation studies conducted on the graphical "know-how" it acquires. Next, the central section of the paper describes a series of questionnaire studies investigating the apprenticeship metaphor. The paper concludes with an evaluation of these experiments and their ramifications for studying knowledge acquisition systems in general.

2 Review of Metamouse System

Our system for acquiring graphical concepts works within an experimental 2D interactive picture editor. The Metamouse is a graphical robot, similar to the LOGO turtle [Papert 80], that follows the teacher through a task, highlights spatial relations it observes before and after each action, and carries out predicted actions. The brief description below summarizes that given in [Maulsby 89].

The learning system, illustrated in Figure 1, induces procedural knowledge from action traces supplied by the user. It includes generalization modules that induce variables and isolate critical constraints from observed actions and spatial relations. It creates a state-graph representation of a program by matching actions and coalescing states, as in [Andreae 85]. The system includes a constraint solver to perform actions described by the nodes in the graph.

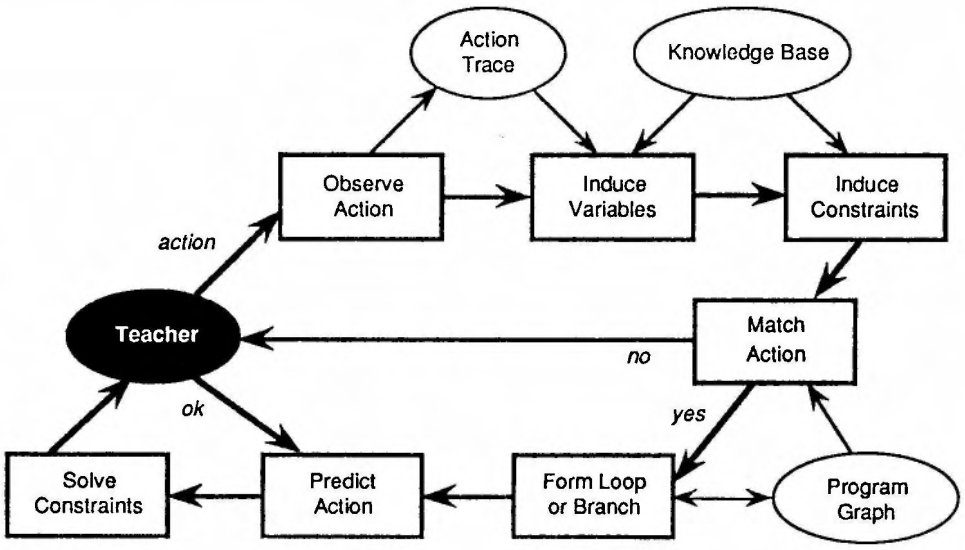
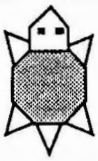


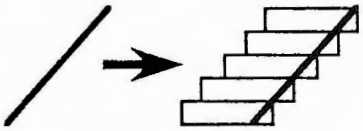
Figure 1. Main components and data flow of learning system.

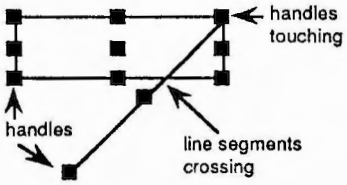
Interaction between teacher and system comprises interleaved sessions of demonstration by the teacher and performance by the apprentice. As the teacher carries out an atomic action, Metamouse follows the cursor and highlights touch relations between objects encountered. If the action appears to be unconstrained, Metamouse interrupts to ask the teacher for an explanation (one of a small set of standard responses). If the teacher's action matches a step in the program graph, the system attempts to predict subsequent actions — in fact, successful prediction is used to confirm the match. The teacher can reject a prediction, in which case alternatives are tried. If no performable actions remain, the system asks the user to teach it what to do next, thus completing an interaction cycle.



My name is Basil and as you can see I'm a turtle. I'm here to help you draw. You teach me repetitive and finicky tasks — like evenly spacing a row of boxes or reconnecting a group of lines when one is moved. I learn by acting as your apprentice — I follow you till I think I know what you'll do next, then I try to do it for you. If I guessed wrong, you give me a gentle tap so I'll undo it and wait for you to show me what's right. I'm eager but don't worry I only predict after I see you do something you've already taught me.

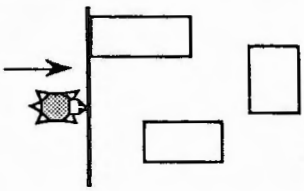
I can draw lines and boxes and carry them by their handles (which I grasp with my jaws). You can teach me to make tools for a task — for example to build a staircase of boxes, use a diagonal line. When done with a tool, delete it so I learn to clean up after the job.

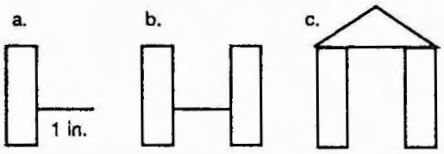




Although I have a good memory, I don't see too well. Because I crawl around a video screen I see things edge on — that makes it hard to spot patterns. Instead I work mainly by feel. I remember how things fit together, which parts — handles and line segments — are connected. In building the staircase I could learn to copy a step by tracing over it, so all the handles of the new step mate with the old. Then move the new step until it sits on top of the old and its lower-right corner touches the diagonal line.

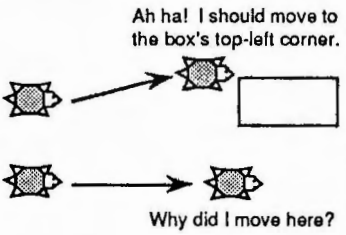
I'm touch-sensitive only at my snout but I can sense contact between what I'm grasping and anything else it touches. If I have to find, say a box, I set off in the general direction you've taught me (up, down, left, right) until I bump into one. It doesn't have to be dead ahead. If you want me to be more selective, give me a tool to carry and teach me to move until it touches something. I'll remember exactly how it touched.





Now, this is very important. I can't learn directly how things should not touch — I mean how they should be separated. You should give me tools to separate them. Say you're drawing an arch and want the columns an inch apart. Draw a one-inch horizontal line, then place the columns at either end of it, as shown on the left.

I only pay attention to what's visible (for me, touchable) on the screen. I don't guess at what you have in your imagination. If I don't sense a bump at the end of an action, I'll ask you to explain why you stopped where you did — whether I should always stop there, or always let you show me where to stop. This is very helpful when making tools that measure — some are constant, some need to be varied each time they are used. If you answer no to both questions, I'll ask you to make a tool that shows why we stopped there.



When you want to teach me, choose "Time for a lesson!" from the Basil menu, and "End of lesson" when you're done. If you want to interrupt the lesson for something else, like working out a method before showing me, just say "Take a nap" — then "Wake up, Basil!" when you're ready. When you don't agree with what I do, just tap me and I'll undo it. In any case when I don't know what to do next, I'll ask you to show me.

So in general you teach me by doing the task yourself, using some extra tools to help me see patterns by feel. As soon as I can predict what to do, I'll take over the task, but I'm always ready to learn something new.

Well that's all I have to say for myself. Hope you enjoy teaching me.

Figure 2. Description of Metamouse given to teachers.

The rules for inducing variables and constraints are too complex to be explained to potential users. For example, spatial constraints are ordered by degrees of freedom they allow: a *determining* constraint selects a particular point on a certain part of a known object; a *strong* constraint may leave the choice of part or object free; a *weak* constraint leaves at least two degrees of freedom. The metaphor explains the system's behavior; it is summarized for the user in terms of Metamouse's sensory system and memory. The metaphor is presented to potential users in a brief written description, shown in Figure 2. Further understanding comes with practice.

3 Validating the Knowledge

Although this paper is concerned primarily with evaluating interaction, it is important to establish that the system actually learns procedural descriptions of concepts. As in the "sloppy modeling" paradigm [Wrobel 88], procedural knowledge is acquired and refined incrementally. The teacher typically demonstrates only as much as is needed to solve the problem at hand. Thus a concept cannot be declared valid or invalid, but its coverage, robustness and complexity may be examined during teaching sessions. Coverage is measured as the ratio of actions correctly predicted to the total number performed (by both teacher and apprentice). The rate of learning is measured as the increase in this ratio from one trace to the next, or between iterations of a loop. Robustness can be measured as the ratio of incorrect predictions (ie. ones rejected by the teacher) to total predictions. Complexity is related to the number of edges in the program graph.

Pilot validation studies have been performed for a number of object-definition, constraint-specification and transformation tasks. The examples were almost, but not quite, noise-free; none contained more than two extraneous or erroneous actions by the user. It was found that the system quickly achieved competence and constructed parsimonious models. The rest of this section describes three such tasks and the results of validation studies.

Performance of the learning system was measured over several traces of each task; the results are presented in Table 1 below. For each trace, we counted the total number of drawing program actions (where an action results in a new resting position for the turtle icon), the number of these actions successfully predicted and performed by Basil, and the number of Basil's predictions that the user rejected. The fifth column shows the completeness of Basil's concept with respect to the current task trace. The final two columns give the number of edges currently in the program graph and the number added during the trace.

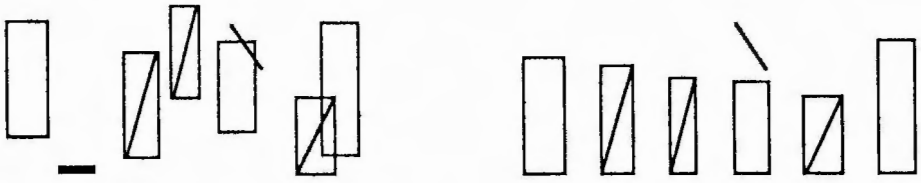
Task		Steps Performed in Trace				Edges in Graph	
	Trace #	Total	# – by Basil – %		Rejected	Total	Growth
Picket-fence	1	29	17	58 %	1	15	15
	2	23	23	100 %	0	15	0
Connectivity	1	6	0	0 %	0	7	7
	2	6	6	100 %	0	7	0
	3	4	1	25 %	1	11	4
	4	4	4	100 %	0	11	0
	5	6	6	100 %	0	11	0
Box-to-line	1	20	8	40 %	0	13	13
	2	24	24	100 %	0	13	0
	3	20	20	100 %	0	13	0

Table 1. Performance of learning system on three tasks.

The goal of the Picket-fence task, shown in Figure 3, is to align a group of boxes and space them at equal intervals. Some of the boxes are marked with diagonal lines, which must be moved to the boxes' new locations. As indicated in Table 1, Basil correctly performed 58% of the actions of the

first trace, since he learned to iterate over the set of boxes, and to handle boxes with and without diagonal lines through them.

When demonstrating the Picket-fence task, the teacher creates a line whose length represents the desired gap between boxes. She then picks the leftmost box and drags it to the left end of the spacer line. She picks the next leftmost box and drags it to the right end. Basil distinguishes the line's end-points from each other and thus does not match these actions. The teacher picks the stranded diagonal line and reattaches it to its box, then moves the spacer line to the box's right edge. When she picks the next leftmost box, which has a diagonal line through it, Basil matches this action with one already learned and conjectures a loop. To confirm it, he predicts the next action, dragging the box to the spacer line, and the teacher assents.



a. Boxes initially scattered; spacer line input by user. b. Result: boxes aligned and evenly spaced.

Figure 3. The Picket-fence task.

Basil predicts the rest of the loop, then attempts the next iteration. The constraint solver scans right for the next box. The nearest one found is rejected because it has no diagonal line. But when Basil skips ahead to the fifth box, which has a diagonal, the teacher rejects his action and picks the fourth instead. Basil creates a program branch to handle this case and records what the teacher does with this box. When she drags it to the spacer line, Basil recognizes this action but is unable to predict its successor in the graph, namely to grasp the stranded diagonal, since there is none; hence Basil continues recording. When the teacher advances the spacer, Basil again recognizes her action and this time is able to predict its successor — to grasp the next box. Thus the branch for handling plain boxes rejoins the main program. Basil successfully predicts all remaining actions until he runs out of boxes; when the constraint solver reports failure, a program branch is necessitated. The teacher takes over, deletes the spacer and declares the lesson over; the branch terminates the loop.

The Connectivity task is shown in Figure 4; after one edge of a polyline is moved, stranded edges are re-connected to it. As a variant, only one vertex is moved. Since the task contains no loops, Basil predicted 0% of the first trace, but induced a program general enough to cover any way of moving the edge, despite incidental contacts. His performance dropped to 25% when the variant was introduced, but thereafter he was able to perform either version of the task.



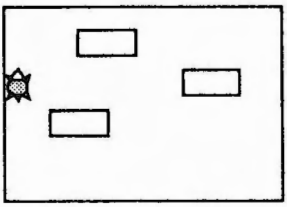
a. User moves one edge of polyline. b. Variant: user moves one vertex.

Figure 4. Connectivity of a polyline.

When the teacher translates the edge (or vertex), the constraint inducer reports a lack of determinacy; Basil interrupts the teacher to ask for an explanation and she chooses the reply "Always ask the user [where to place the object]." Variables in touch relations correctly associate vertices that are to be re-connected. When the variant task is introduced in a subsequent trace, Basil creates a

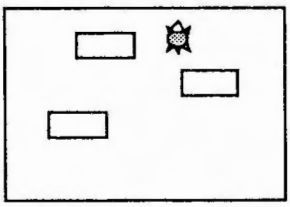
branch for grasping and moving a vertex, but has no criterion for choosing one branch over the other on subsequent traces; this decision must (as it should) be made by the user.

0.



Starting Position.
Teacher has selected "Time for a Lesson" from the Basil menu.

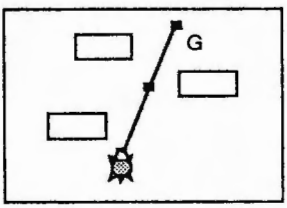
1.



Teacher places anchor point of guideline.
Basil moves to position, asks why; Teacher answers that point is always given by the user, ie. is an input parameter.

```
Relevant constraints:
  position  input
```

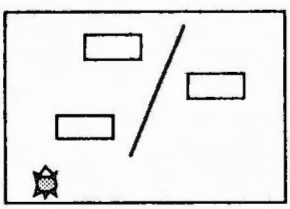
2.



Teacher draws guide line.
Basil requests constraint for placement of end-point; Teacher classifies it an input, as in step 1.

```
Relevant constraints:
  position  input
```

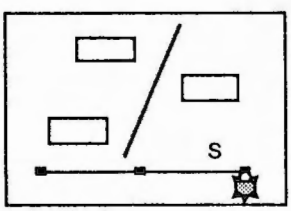
3.



Teacher places anchor pt for sweep line.
Basil requests constraint; Teacher replies position is constant.

```
Relevant constraints:
  position  constant
```

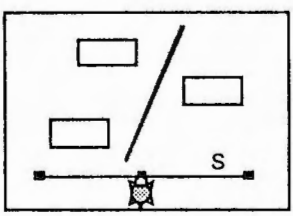
4.



Teacher draws sweep line;
Basil requests constraint; end-point is constant.

```
Relevant constraints:
  position  constant
```

5.

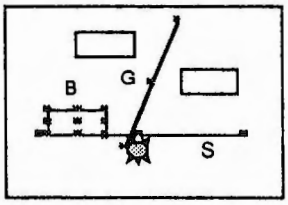


Teacher places cursor at mid-point of S, in order to drag it.
Basil follows.

```
Relevant constraints:
  grasp (S.midpt)  determining:
                   contact is point-to-point
                   object (S) is known
```

Figure 5 a. Box-to-line task trace, steps 0–5.

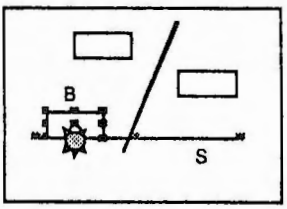
6.



Teacher drags S upwards to contact a box. Basil follows.

```
Relevant constraints:
  touch (S.line: B.bottom)   weak:
                             contact is line-on-line
                             object (B) is found by scanning
  path (upwards)   weak: constrains scan for B
  touch (S.line: G.line)   weak
```

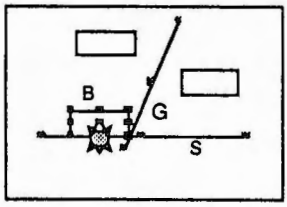
7.



Teacher picks center handle of box B. Basil follows.

```
Relevant constraints:
  grasp (B.center)   determining:
                     contact is point-to-point
                     object (B) is known
```

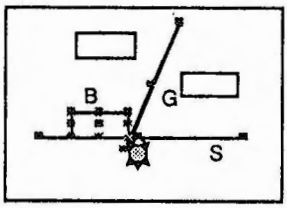
8.



Teacher drags box so that lower right corner touches G.

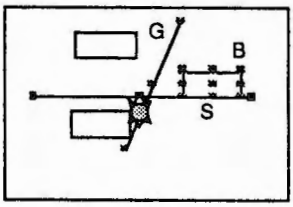
```
Relevant constraints:
  touch (B.bottom.right : G.line)   strong:
                                    contact is point-on-line
                                    object (G) is known
  path (rightwards)   weak
  touch (B.bottom : S.line)  sustained - ignored
```

9.



Teacher picks S, as in step 5.
Basil matches action and relevant constraints,
and tentatively forms a loop from step 8 back to step 5.

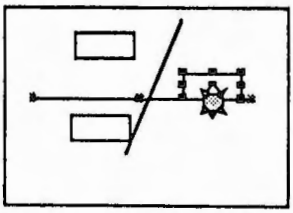
10.



Basil predicts drag S up to a box, as in step 6.
Teacher accepts prediction; Basil confirms loop.

```
Variables updated:
  B set to box found by upward scan
Program modification:
  add step 5 as a successor of step 8
```

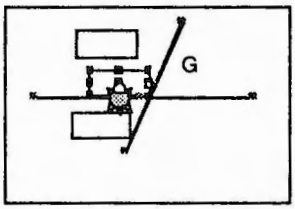
11.



Basil predicts grasp center of box.
Teacher accepts prediction.

Figure 5 b. Box-to-line task trace, steps 6–11.

12.

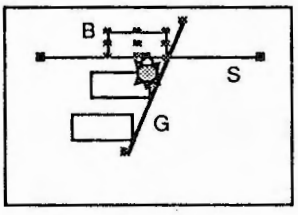


Basil generalizes path in step 8 in order to predict similar action in body of loop.
Teacher accepts prediction.

```
Constraints removed:
  path (rightwards)
```

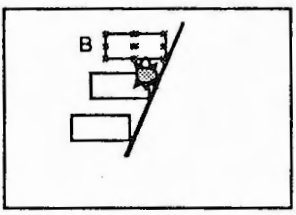
13 ... 16. Basil predicts third iteration of loop; no explicit generalization rewrites required.

17.



Basil predicts re-entry of loop at step 5, ie. pick S at midpt.
Teacher accepts prediction.

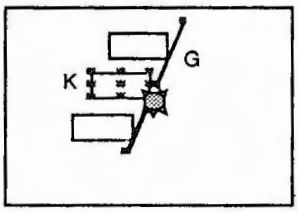
18.



Basil unable to perform next step of loop, ie. drag S up to contact some box.
Teacher resumes control; deletes S.

```
Program modification:
  add alternative action to successors
  of step 5
```

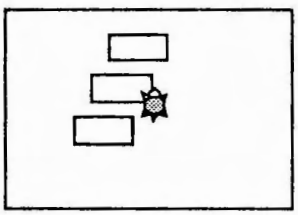
19.



Teacher picks G at midpt.
Basil follows.

```
Relevant constraints:
  grasp (G.midpt)  determining:
                   contact is point-to-point
                   object is known
```

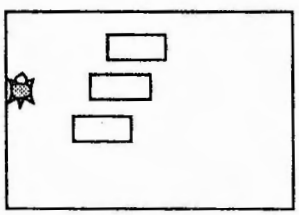
20.



Teacher deletes G.

```
Relevant constraints:
  none on postconditions
```

21.



Teacher signals end of lesson.

```
Program modification:
  add "stop" as successor of step 20
```

Figure 5 c. Box-to-line task trace, steps 12–21.

The object of the Box-to-line task is to translate boxes horizontally so that their lower-right corners are aligned. Basil learned the loop during the first trace and performed 40% of the actions. The number and initial position of the boxes were varied during the subsequent traces, producing some coincidental touch relations that were already covered by the constraints induced from the first example.

A teaching trace is presented in Figures 5 a–c. Note that the teacher's and Basil's actions are described at the top-right corner of each frame. Notes printed in `Courier` typeface indicate learning system activities such as isolating important constraints and forming loops and branches. The program graph induced from this example execution is illustrated in Figure 6.

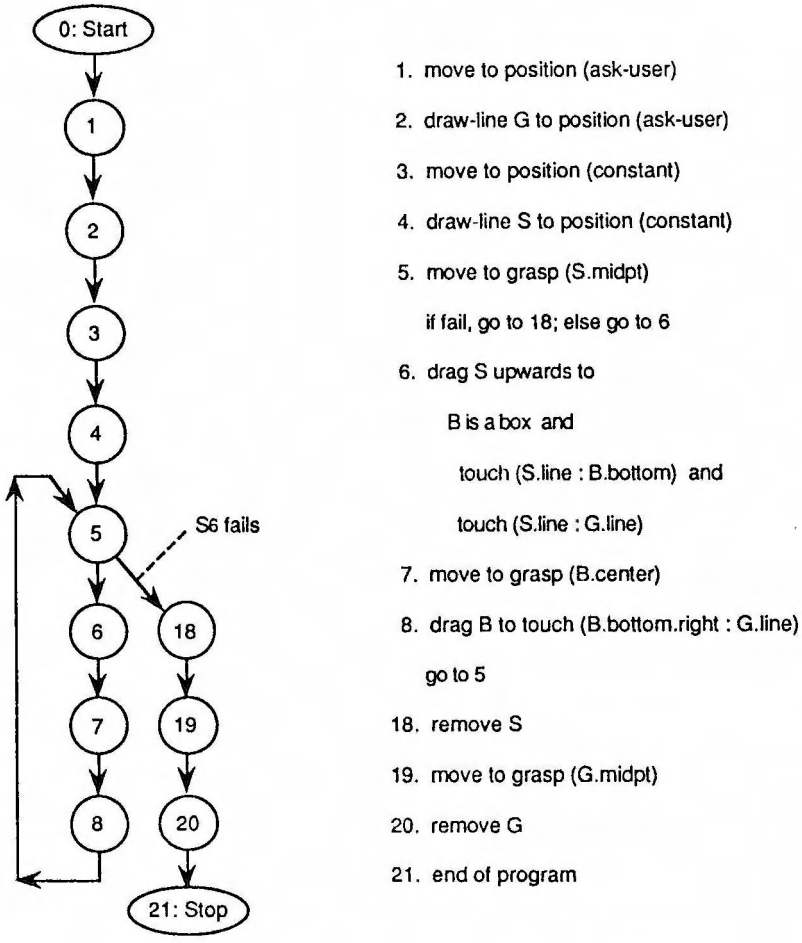


Figure 6. Program graph induced for Box-to-line task.

4 Evaluating Interaction

A critical aspect of a learning system is that the teacher understand its behavior [MacDonald 87]. Users of the system are given a metaphor — Basil the Metamouse — to help their understanding. Its suitability, measured as the ease with which teachers learn to predict its behavior, can be studied apart from a working system. Two questionnaire-based experiments are described here. The first, a pilot study without controls, was intended to establish the viability of a questionnaire and obtain rough estimates of the metaphor's comprehensibility. The second introduced controls on the amount of

prior knowledge subjects had regarding the metaphor, and also measured correlations with previous computing experience. The results of both experiments suggest that Basil's behavior is largely self-explanatory or easily rationalized. They do not show that users can easily create procedures by teaching Basil.

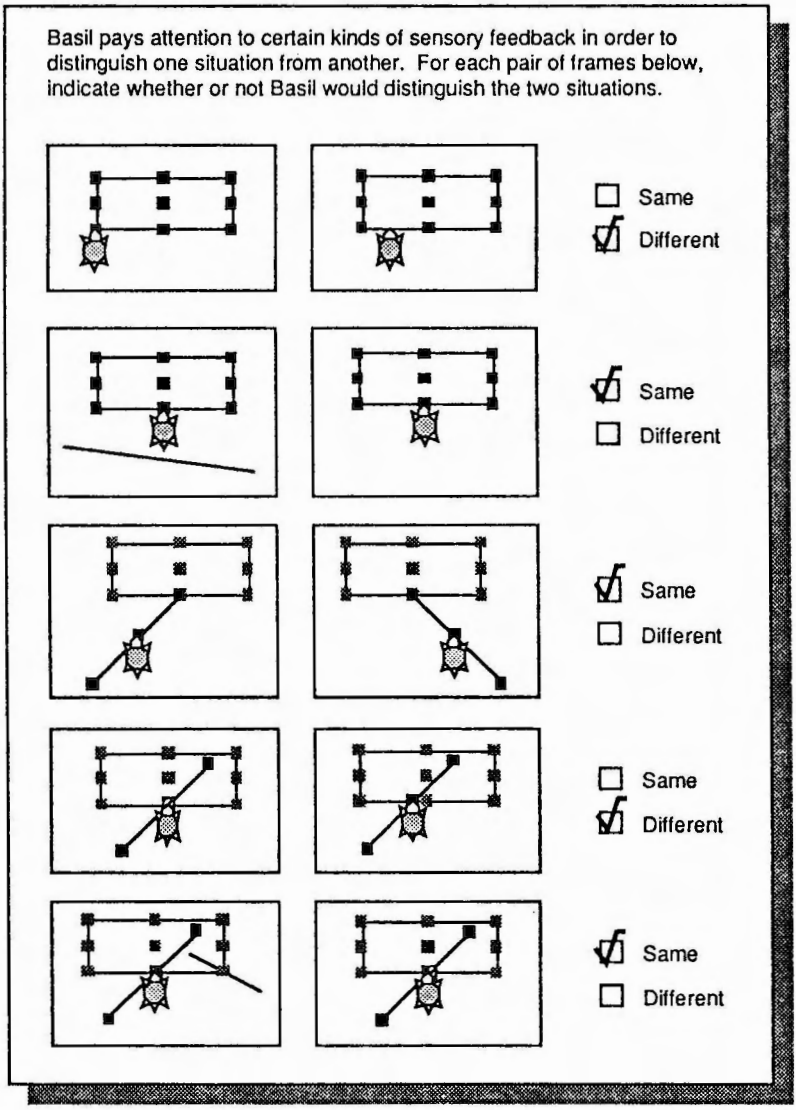


Figure 7. Sample page from questionnaire, with correct responses marked.

4.1 Pilot Study

In the pilot study, a number of potential users were given the brief description of Basil shown in Figure 2 and then asked to work through a self-study guide. Typical questions depict a situation and ask the subject to predict Basil's response. Scores on each page of questions were recorded separately so that progress could be measured. The study guide provides correct answers after each page, to simulate system feedback. Subjects were asked not to refer back to previous questions, however.

The study guide contains approximately 55 questions, of several types:

- two graphical situations depicted (Basil in contact with objects); subject asked whether Basil matches these situations to each other (approx. 30 questions)
- two actions depicted, showing situations before and after; subject asked whether Basil matches these actions (approx. 20 questions)
- a sequence of actions; subject asked to differentiate those performed by teacher from those performed by Basil (1 question)
- a graphical task is specified; subject asked to create an algorithm that Basil could learn (3 questions).

The questions are arranged in the order given above, so that difficulty tends to increase as subjects become more experienced. A sample page is shown in Figure 7: the subject is asked to state the discriminations he or she believes Basil would make between touch relations. The experiment was run with eight volunteer subjects, all computer scientists. All were allowed to work at their own pace at a time of their choosing.

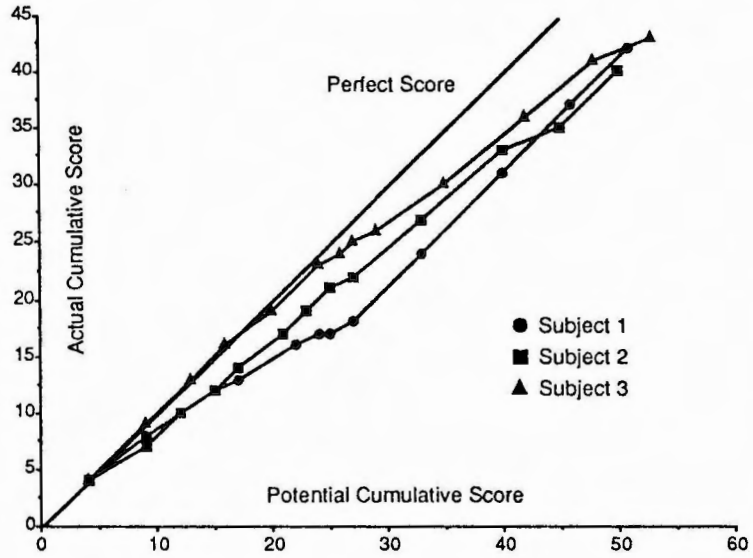


Figure 8. Performance of three typical subjects on Basil questionnaire.

The first three groups of questions were graded for use in this study. Subjects' understanding of Metamouse at any point is measured as the ratio of the number of questions correctly answered to the total questions counted. This ratio is plotted for three subjects in Figure 8; perfect performance lies along the line of slope 1. An increase in slope represents improvement in performance.

If the metaphor were difficult to understand, one would expect numerous errors in the early sets of questions, with at best a slow improvement. But if completely obvious, one would expect near-perfect performance from the beginning with no degradation. The actual plots maintain a fairly steady slope with some inflections. They show near-perfect performance initially, with occasional mistakes and difficult spots after which near-perfect performance is restored. This suggests that the "superficial" aspects of the metaphor — namely the rules that distinguish parts of objects and types of direct touch — are easily understood, while deeper aspects — the rules that govern action-matching and prediction — are less well understood but learnable.

A number of methodological deficiencies prohibit stronger conclusions from this experiment. One particularly thorny issue is this. All of the subjects commented that a number of the questions they answered correctly were difficult. Some questions were marred by photoduplication and have been eliminated from the data. Others however have been buried in the numerical results because the difficulty of individual questions was not measured. Since the number of questions in each sample set is small, this masks the degree of difficulty one is trying to measure and lessens the significance of the results and the utility of a (necessarily small) questionnaire as opposed to real interaction.

4.2 Controlled Study

Based on performance and written comments from the subjects of pilot study, we concluded that the questionnaire was a reasonable vehicle for further experiments. We decided to investigate two hypotheses. First, if the metaphor is intuitively appealing, the amount of explanation given prior to interacting with it will not significantly affect performance. Thus, subjects given little or no explanation would learn as well as those given the manual (Figure 2) as they worked through the questionnaire. The second hypothesis is that background experience with computer systems will not significantly affect performance. The first hypothesis was not disproven. The second was contradicted, but it was found that the most useful *types* of experience were prior use of graphical interfaces and drawing programs.

The second study introduced several controls. First, the order of questions was varied, in order to simulate interaction rather than guided study; this was found to have no significant effect on performance. Second, some subjects were given no answer key (ie. no feedback); this control group was eliminated due to lack of response. Third, the introductory material was varied. One version of the questionnaire contained the full description of Basil (Figure 2). A "less informative" version came with a two-page example of Basil learning the Box-to-line task, similar to Figure 5 but without notes regarding the learning system's internal machinations. The minimal version provided a meagre one-paragraph explanation of terms used in the questionnaire. Subjects were asked to rate (on a scale of 0–5) their previous experience in several aspects of computing: a) use of systems with graphical interfaces; b) drawing programs; c) programming; d) spreadsheets and data-bases. Although more than 150 questionnaires were distributed to students of architecture, industrial design and computer science, only 16 were satisfactorily returned.

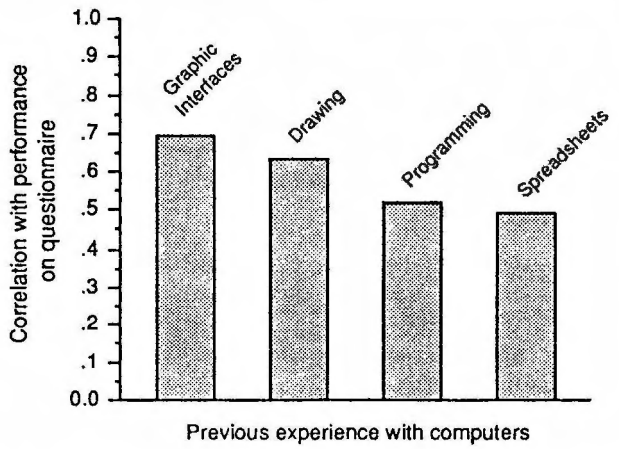


Figure 9. Effect of previous experience on subjects' ability to predict Basil.

In order to test the first hypothesis, the returned questionnaires were pared down to three sets of four, in which the four subjects had comparable reports of previous experience with computers ($F(2,9) = 2.059$, $p = .1835$). There was no significant difference between the three groups, despite

the radical differences in the amount of background information they received concerning the metaphor ($F(2,9(= 1.617, p = .2512$). Thus the first hypothesis was not disproven, lending support to the claim that the metaphor is intuitively appealing.

The analysis showed, however, that performance was significantly correlated with previous aggregated computing experience. Subjects with more experience did much better on the questionnaire. This contradicted the second hypothesis. On the other hand, a breakdown of experience by domain, as in Figure 9 above, shows that having worked with graphical interfaces and drawing programs confers the greatest advantage. This is just the sort of background that users of Basil would have.

5 Discussion and Conclusions

The rate at which potential users learn to understand the behavior of Basil the Metamouse, given a brief introductory description, was measured as the improvement in their ability to predict Basil, that is, to answer questions about what he would do and what distinctions he would make. Learning curves (eg. Figure 8) indicated that the subjects of our studies had a good understanding from the start, and their performance neither improved nor deteriorated significantly as subsequent questions became harder. Furthermore, although groups of questions isolated aspects of Basil's behavior, performance did not vary greatly from one to another. The conclusion of these preliminary experiments is that the metaphorical apprentice with a limited sensory system and a notion of conditional action is comprehensible to potential users.

Such experiments may lead to modifications of the user interface. In fact, difficulties experienced by subjects of the pilot experiment in predicting Basil's sensory discriminations and classification of constraints motivated a more telling graphical representation for Basil's sensory feedback [Maulsby 89].

Ramifications of such interface studies for knowledge acquisition focus on the deeper aspects of elicitation systems. In particular, they facilitate a detailed consideration of the metaphors — express or implied — in which such systems involve their users through their interaction techniques. The questionnaire method isolates aspects of interaction and may reveal distinctions and assumptions that system designers should reconsider.

References

[Andreae 85]
Andreae, P. M. "Justified generalization: acquiring procedures from examples." PhD dissertation. Department of Electrical Engineering and Computer Science, MIT. January 1985.

[Benbasat 88]
Benbasat, I. and Dhaliwal J. S. "A framework for the validation of knowledge acquisition." *Proc. 3rd Knowledge Acquisition for Knowledge-Based Systems Workshop,* pp. 1.1-1.18. Banff AB CDN. November 6-11, 1988.

[Fuller 88]
Fuller, N. and Prusinkiewicz, P. "Geometric modeling with Euclidean constructions," in [Thalmann 88], pp. 379-391.

[Gruber 87]
Gruber, T. R., and Cohen, P. R. "Design for acquisition: principles of knowledge-system design to facilitate knowledge acquisition." *IJMMS* 26(2), pp. 143-159. February 1987.

[MacDonald 87]
MacDonald, B. A. and Witten, I. H. "Programming computer controlled systems by non-experts." *Proc. IEEE SMC Annual Conference.* Alexandria VA. October 1987.

[Maulsby 88a]
Maulsby, D. L. and Witten, I. H. "Acquiring graphical know-how: an apprenticeship model." *Proc. European Knowledge Acquisition Workshop (EKAW '88).* Bonn. June 1988.

[Maulsby 88b]
Maulsby, D. L. "Inducing procedures interactively." MSc thesis. Research report no. 88/335/47. Department of Computer Science, University of Calgary. December 1988.

[Maulsby 89]
Maulsby, D. L., Kittlitz, K. A. and Witten, I. H. "Metamouse: specifying graphical procedures by example." *Proc. SIGGRAPH '89* (in press).

[Noma 88]
Noma, T., Kunii, T. L., Kin, N., Enomoto, H., Aso, E., Yamamoto, T. Y. "Drawing input through geometrical constructions: specification and applications," in [Thalmann 88], pp. 403-415.

[Papert 80]
Papert, S. *Mindstorms*. Basic Books. New York. 1980.

[Shaw 88]
Shaw, M. L. G. and Woodward, J. B. "Validation in a knowledge support system: construing and consistency with multiple experts." *IJMMS* 29(3), pp. 329-350. September 1988.

[Thalmann 88]
Magnenat-Thalmann, N., Thalmann, D., eds. *New Trends in Computer Graphics: Proc CG International '88*. June 1988.

[Wrobel 88]
Wrobel, S. "Design goals for sloppy modeling systems." *IJMMS* 29(4), pp. 461-477. October 1988.

A General Approach to the Use of Background Knowledge in a Numerical Induction Algorithm

Werner Mellis
Nixdorf Computer AG
Pontanusstr. 55, D-4790 Paderborn
email: mellis.pad@nixpbe.uucp

Abstract: Top down induction of decision trees is an interesting method for generating rules for knowledge bases of expert systems. But up to now its diverse realisations (most prominantly ID3) are restricted to a simple attribute value language. Even more restrictive is its ignorance of any background knowledge. In this paper a semantical analysis of ID3 is presented, which leads to a generalised procedure capable of using background knowledge and a more expressive language.

1. Introduction

When we started to develop a case data based knowledge acquisition component for an expert system shell, we assumed that a huge set of case data carefully prepared for carmatic knowledge acquisition will be provided by the applier. Therefore it seemed obvious to choose an adaption of ID3 (Quinlan 1983) as the starting point of the development of the Nixdorf induction algorithm NINA. During development and application we have basically been confronted with three different types of problems. 1. The generated rules should profit from the expressive richness of the shell's rule language. 2. The system should generate readable rules. 3. It is inappropriate to assume the existence of a huge set of carefully prepared case data. In applications the case data will not be carefully prepared in the sense of being adapted to the induction algorithm. On the contrary the induction algorithm has to be adapted to the case data's deficiencies and peculiarities.

The mentioned problems have first been conceived as quite different problems. However, a thorough analysis pointed to a common origin. All these problems are a consequence of the purely syntactic nature of ID3. It does not make use of any background knowledge expressing semantic relations between the attributes. Investigations of applications prove, that it is often very easy to formalise the appropriate background knowledge. But it also turned out to be rather difficult to integrate a useful and correct processing of this knowledge into the ID3 algorithm. The solution finally arose from a careful logical analysis of the ID3 algorithm. The logical analysis and a logically sound and efficient way to use background knowledge in a machine learning tool, which is designed as data analysis environment is discussed in this paper.

Paragraph 2 describes some problems and their solution using background knowledge. In paragraph 3 we generalise the expansion rules of ID3 leading to a generalised induction procedure capable of using

background knowledge. In paragraph 4 we present a logical analysis of the generalised procedure and demonstrate, how background knowledge is related to the generalised expansion rules. In paragraph 5 we discuss how generalised expansion rules can be generated from background knowledge represented in predicate logic.

The problems appeared in applications as different as for example configuration of automatic cashier systems and classification of dialects. To avoid explanation of the many complicated technical details of the real applications, simple artificial examples are used to ease explanation of the logical problems.

2. Examples of Background Knowledge

2.1 Incomplete case data and redundancy in rules

Often case data are incomplete, because for some cases the values of some attributes are not given. The recording person just assumed some obvious facts:

```
species(X,mammal)       -> type(X,animal)
species(X,cat)          -> size(X,small)
species(X,elephant)     -> size(X,big)
```

Let us assume, that the appropriate completion of the case data has been accomplished and a decision tree has been developed. To convert the tree into a set of rules every branch is described by a rule. The rules generated from the case data may not be satisfying:

```
if type(X,animal)
and species(X,mammal)
and size(X,small)
then class(X,pet)
```

The first conjunction element of this rule is superfluous. Because of:

```
species(X,mammal) -> type(X,animal)
```

The rule is equivalent to:

```
if species(X,mammal)
and  size(X,small)
then class(X,pet)
```

However, rule optimisation of this kind is not always wanted. Sometimes redundancy in rules is useful. In goal driven rule application user interaction is often controlled by the order of premisses. An expert system containing the rule:

```
if sex(X,female)
and  pregnant(X)
then prohibited(X,smoking)
```

will under certain conditions ask first for sex and then for pregnacy. On the contrary the optimised rule will first ask the more specific question for pregnacy, but this will have a rather irritating effect on a male patient, especially if later the system asks for the patient's sex. Of course this problem should better be solved by explicit control of the user interaction. But many systems do not offer such possibilities. Using background knowledge in induction should make appropriate control of rule generation possible.

2.2. Additional Concepts

Many investigations in machine learning demonstrated that the concepts used in case data are often not well suited for the formulation of general hypotheses. If the ID3 algorithm is used in such cases, it will successfully generate a decision tree, but its branches do not represent interesting rules about the domain of application. To produce interesting rules, concepts not occuring in the case data have to be introduced into the rule language. As it is very difficult to automatically generate new concepts, at least a manual support for their introduction is necessary. Of course one might demand a reformulation of the case data using more appropriate concepts. But if an induction system is to be used as a data analysis system, it ought to support introduction of new attributes without changing the case data. An approach to support this is to allow the definition of new concepts as part of background knowledge used by the induction algorithm. By extension of the background knowledge the induction system can then be adapted to the analysis of data in various ways.

Two examples of background knowledge:

(1) species(X,puppy) <- species(X,dog),age(X,joung)

(2) species(X,carnivore) <- species(X,lion) v species(X,tiger)

The first clause allows to combine two attributes. Instead of the rule

```
if species(X,dog)
and age(X,joung)
then class(X,pet)
```

a simpler rule can be generated

```
if species(X,puppy)
then class(X,pet)
```

The second clause introduces a wider concept, which allows to comprehend several values. Instead of the rule:

```
if species(X,lion)
and material(X,textile)
then class(X,toy)

if species(X,tiger)
and material(X,textile)
then class(X,toy)
```

a simpler rule can be generated:

```
if species(X,carnivore)
and material(X,textile)
then class(X,toy)
```

2.3. Relations in Premises

Let us assume, that the relation bigger(X,Y) is defined in the usual way. There are two possibilities to use this definition for rule induction. As described above, one possibility is the introduction of an auxiliary attribute for the comparison of the two attributes attr_a and attr_b.

```
a_bigger_than_b
        <-> attr_a(X,A),attr_b(X,B),bigger(A,B)
```

More interesting is an introduction by the generation of hypotheses. To achieve this, the process of hypothesis generation has to be sufficiently powerful. In ID3 hypothesis generation is a stepwise process of refinement of some initial hypothesis. A hypothesis is a rule, whose premise is a conjunction of elementary premises of the form attr(X,a), where X is a variable and a a value of the attribute attr. Each refinement step consists in the addition of a new elementary premise related to an attribute not yet occuring in the premise.

There is of course always the possibility of ad hoc extensions of the hypothesis generation to allow for the introduction of specific elementary premises like attr(X,A) & A > limit etc.. In this case the ID3 algorithm basically proceeds as before. The set of all case data is decomposed according to the new separation

attr(X,A) & A > limit , attr(X,A) & A =< limit.

The information gain is compared as usual with that of alternative separations and the best is selected. The only difference is, that there is a larger selection of separations to choose from.

It is not necessary to adapt the hypothesis generation by ever new ad hoc extensions. Rather one can rely on a general logical mechanism for the generation of separations and a general adaption of the procedure to use them. This mechanism not only produces separations like

attr(X,A) & A > limit , attr(X,A) & A =< limit

but also recognizes this to be coarser than the separation

attr(X,A) & A > limit, attr(X,A) & A = limit,
attr(X,A) & A < limit.

3. The Use of Background Knowledge in the Induction Procedure

Given a set of classified examples a numerical induction procedure like ID3 recursively constructs a decision tree. To each node of the tree it associates a set of examples and to each link an equation. In each recursion step a node n of the tree, is considered for expansion. An attribute a, not occuring in an equation associated to the branch above the node, is selected. If k different values of the attribute a occur in the examples, the node n is expanded by k links. To each link one of the equations $a=v_i$ is attached. To complete the expansion step the set of examples associated to the node n is separated. The set of all examples satisfying the equation $a=v_i$ is attached to the new node at the link, to which the equation $a=v_i$ is associated.

There are different criteria to terminate the expansion of the tree and different ways to determine a value for the leafs of the tree. To reduce the search effort the selection of attributes is according to some statistical evaluation. For each node only one attribute is selected for expansion. Because these aspects are not of interest for our present purpose, we avoid describing them in detail.

The completed tree can be used for classification of new examples. The example is moved down the tree until it reaches a leaf. The value at the leaf represents the class into which the example is classified. The example can only move down a link if it satisfies

the equation attached to it. Since the equations attached to the links leaving a node cannot be satisfied simultaneously by an example, each example travels a uniquely determined path down the tree. The tree can therefore also be described as a set of rules. Each leaf node is taken as the conclusion of a rule, whose premise is the conjunction of all the equations attached to the branch leading to the leaf node. In this respect we may call the tree cummulative.

In order to make use of background knowledge in the induction procedure we generalise the decision tree and the expansion steps used to build it. This generalision does not touch the declarative semantics of the tree. But it changes its procedural semantics and makes it non-cummulative.

We describe the expansion step of the generalised procedure by rules like: $C,P \dashrightarrow f_1,\ldots,f_k$, where C,P and $f_1,\ldots,f_k$ are formulae. This rules are used in the following way: If the "condition associated to a node n" is C and C contains P as a conjunction element, then expand n by k links to new nodes $n_1,\ldots,n_k$ and associate f_i as an "add condition" and P as a "delete condition" to the i-th link. The "conditions associated" to the node n_i is the conjunction of the add conditions, which do not occur as delete conditions above n_i. $f_1,\ldots,f_k$ must represent a complete separation, i.e. for each example exactly one of the formulae must be satisfied.

The ID3 expansion step can be described by such rules either. The rules to describe ID3 are special cases, where P is empty and the formulas f_i are equations $a=v_i$, where a is an attribute and v_i is a value for a occuring in the examples. It is obvious that such rules build up a cummulative tree.

The general rules allow to use a more expressive language. That this is possible with an ID3-like procedure has already been noticed by Bundy in [9]. However, the expansion rules we allow here are still more general, as they also allow to use some background knowledge.

Consider for example the rule:

(1) C,genus=mammal --> species=dog,species=cat.

It expresses a hierarchy: the concepts dog and cat together form a complete separation of the concept mammal. This can be used to avoid overspecialisation of rules. We assume, that without rule (1) the procedure generates the overspecialised rules

```
if species=dog
then class=pet

if species=cat
then class=pet
```

using the ID3 expansion rule

(3) C,empty --> species=dog,species=cat.

Introducing the hierarchy of mammal, dog and cat the procedure can apply the two generalised expansion rules

(1) C,genus=mammal --> species=dog,species=cat.

(2) C,empty --> genus=mammal,genus=reptile.

If we apply expansion rule (1) only if genus=mammal is contained in C, expansion rule (2) is used before (1) generating the rule:

```
if genus=mammal
then class=pet
```

which summarizes the two rules above.

Only little corrections have to be made to the ID3 procedure to achieve this. The procedure has to employ the generalised expansion rules instead of the special expansion rules for ID3. Further every formula on the right hand side of an expansion rule has to be decidable for every case datum. Therefore in general the procedure has to make use of a theorem prover to determine validity of a formula with respect to a given example.

In this way background knowledge can be expressed by expansion rules. But one may doubt whether this is appropriate. We therefore will continue to investigate the possibility to express background knowledge as statements in first order predicate logic. There are two questions we are interested in. 1. Is there a more general exploitation of background knowledge than the use of expansion rules? 2. Is there a more appropriate representation of background knowledge?

4. Expressing Background Knowledge in First Order Predicate Logic

4.1 Complete Separations and Expansion Rules

In the following the well known terminology of predicate logic and logic programming is used without further explanation. In general it is refered to [3] and [4].

The input of ID3 are descriptions of case data, within which a certain literal is distinguished as the one to be concluded.

Definition:

An unclassified case data is a triple $Fu=(L,\alpha,\theta)$, where L is a set of literals, α the set of free variables of L and θ is a variablefree substitution for α.

A classified case data is a pair $F=(A,(L,\alpha,\theta))$, whose first element is a literal, whose free variables occur in α and whose second element is an unclassified case data.

Example:

```
F=(class(X,pet),
  ({type(X,animal),genus(X,mammal),species(X,cat)},(X),(X/fritz)))
```

For each case datum the induction procedure finds a suitable clause covering the case datum. This is a clause, which allows to conclude the distinguished literal provided, that the right interpretation (one in which the classified case datum holds) is assumed. This means, that one has to demand, that the classified case datum holds in the interpretation and that the clause is satisfied. This reasoning leads to a concept similar to those of Shapiro (Shapiro 1983) and Buntine (Buntine 1986).

Definition:

A clause A <- $B_1,...,B_n$ covers the classified case datum $F=(A',(\{A_1,...,A_m\},\alpha,\theta))$ in the interpretation I, if firstly $A'\theta$, $A_1\theta,...,A_m\theta$ hold in I and if secondly there is a substitution σ, which unifies A and $A'\theta$ and for which $(B_1\&...B_n)\sigma$ holds in I.

Example:

classified case datum:

```
F=(class(X,pet),
  ({type(X,animal),genus(X,mammal),species(X,cat)},(X),(X/fritz)))
```

Let I be an interpretation, in which

```
      for all X: species(X,cat) -> size(X,small),
      for all X: genus(X,mammal) -> type(X,animal),
      genus(fritz,mammal),
      class(fritz,pet) und
      species(fritz,cat)
```

hold. Then the clause

```
      if genus(X,mammal)
      and  size(X,small)
      then class(X,pet)
```

covers F in I.

As ID3 like algorithms at first build up the premises for the desired rules and only at the end add suitable conclusions, the introduced concept will not be used right now.

It is characteristic for ID3 like induction that several concepts are induced at once. In the given example one might imagine, that beside the class pet also such classes as fear producer are induced.

This kind of induction can be described as searching the space of complete separations. Roughly said, a complete separation is a set of formulae satisfying the disjointness and a completeness condition. A set of formulae is disjoint, if none of them can be satisfied by one example simultaneously with any other formula of the set. A set is complete, if any example satisfies one of the formulae. (This notion will be precisely defined later.) To understand this the decision tree generated by ID3 like algorithms is represented by its branches. Each branch is a condition, more exactly a conjunction of elementary conditions. The conditions are disjoint and complete. This means any possible case datum satisfies exactly one condition. The conditions therefore constitute a complete separation. In case not all of the (unclassified) case data satisfying one condition have the same distinguished literal (i.e. belong to the same class), the condition is replaced by some more specific conditions, such that the conditions together form again a complete separation, which is now more fine grain. The goal is to find a coarsest complete separation, such that only case data with the same distinguished literal fall into one class. If such a separation is found the conditions are considered premises of rules with the appropriate conclusions. This yields a set of rules covering the classified case data. (In order to avoid confusion, we should keep in mind, that the description given here applies also to a non-cummulative generalisation of ID3 as described in paragraph 3. Also in the following we will use the name ID3 for both ID3 and its generalisation.)

In order to arrive at a precise description of this intuitive idea, some auxiliary terms are needed.

Definition:

Let ß be a set of variables. A pair $L=(\{B_1,\ldots,B_m\},ß)$ is a condition for ß, if $B_1,\ldots,B_m$ are literals and the variables ß occur in $B_1,\ldots,B_m$.

A condition $L=(\{B_1,\ldots,B_m\},ß)$ for ß covers an unclassified case datum $F=(\{A_1,\ldots,A_n\},\alpha,\theta)$ in the interpretation I, iff ß is a subset of α, $A_1\theta$ &...& $A_n\theta$ holds in I and there is a substitution σ, such that $B_i\theta\sigma$ holds in I.

Remark:

If no confusion is to be expected in the following the set $\{B_1,\ldots,B_m\}$ and the conjunction $B_1\&\ldots\&B_m$ are denoted by L just like the condition itself.

Using the introduced terms the intuitive idea, that the ID3 induction is a search in the space of separations along a refinement hierarchy, can be given a precise definition. Background knowledge is described as a theory.

Definition:

Let $L_i=(\{B_{i1},\ldots,B_{im_i}\},\alpha)$ be a set of conditions and $L=(\{A_1,\ldots,A_n\},\alpha)$ be a condition. With exception of α let all variables of L and L_i be different. The set L_i is a complete separation of the condition L in respect to the theory T, iff for each unclassified case datum $F_u=(M,ß,\theta)$ with α subset of ß and for each model I of T holds

(1) if L covers F_u in I, then there is exactly one i, such that L_i covers F_u in I and

(2) for all i: L covers F_u in I, if L_i covers F_u in I.

The following lemma proves the semantic concept of separation to be recursively enumerable.

Lemma 1:

Let $L=(\{A_1,\ldots,A_n\},\alpha)$ be a condition and $L_i=(\{B_{i1},\ldots,B_{im_i}\},\alpha)$ be a set of conditions. With exception of α let all variables be different. Let $\theta(\tau)$ be a variablefree substitution of L_i (L) with constants not occuring in neither L_i nor L nor T. L_i is a complete separation of L in respect to the background knowledge T, iff

(1) $T \& L_i\theta \Longrightarrow \exists L\theta$ for all i,
(2) $T \Longrightarrow \forall(\neg L_i \vee \neg L_j)$ for all $i \neq j$ and
(3) $T \& L\tau \Longrightarrow \exists(L_1 \vee \ldots \vee L_m)\tau$.

Lemma 1 provides us with a clear understanding of the declarative semantics of expansion rules. Further it is the basis for generating generalised expansion rules. Let us assume that some background knowledge is given as a theory T. If we can prove from T using lemma 1, that L_i is a complete separation of L, then this justifies the use of an expansion rule $C,L \dashrightarrow L_i$.

Finally we have to specify how the constructed separation is turned into a set of clauses covering the classified case data.

Definition:

Let $L_i=(\{B_{i1},\ldots,B_{im_i}\},\alpha)$ be a separation and $K_i=B_{i1},\ldots,B_{im_i} \rightarrow C_i$ a set of clauses. K_i cover the set of classified case data F in respect to the theory T, iff for all case data f_j there is a clause K_i, such that K_i covers f_j in respect to T.

Remark:

The notion of comlete separation is recursively enumerable, but in general (for arbitrary theories) not recursive.

ID3 like procedures search the space of separations for a coarsest separation covering all case data. The search proceedes from coarse to finer separations. It is therefore important to determine the coarsest, proper, complete separations of a given condition.

Definition:

Let $L=(\{A_1,\ldots,A_n\},\alpha)$ be a condition and let $\{(L_i,\alpha)\}$ $(i=1,\ldots,n)$ and $\{(K_j,\alpha)\}$ $(j=1,\ldots,m)$ be complete separations of L in respect to the background knowledge T. With exception of α let all variables be different.

The separation $\{L_i\}$ ist called finer than the separation $\{K_j\}$, iff $n > m$ and there is a surjective mapping $i: \{1,\ldots,n\} \rightarrow \{1,\ldots,m\}$, such that for all $j \in \{1,\ldots,m\}$: $i^{-1}(K_j)$ is a separation of K_j in respect to T.

This definition reduced the problem of refining a separation to finding a separation of a condition. A coarsest refinement of a separation can therefore be found by replacing a condition by one of its coarsest separation.

Lemma 2:

Let $L=(\{A_1,\ldots,A_n\},\alpha)$ be a condition. Let $L_i=(\{B_{i1},\ldots,B_{im_i}\},\alpha)$ $(i=1,\ldots,k,\ k > 1)$ be a complete separation of L in respect to the background knowledge T. With exception of α let all variables be different. Let θ be a variablefree substitution of L_i with constants occuring in neither L nor T.

$\{L_i\}$ is a coarsest (proper) complete separation of L in respect to T, iff there is no set of atoms $B=\{B_1,\ldots,B_r\}$, in which the variables α occur and in which all variables different from α are different from those in L and L_i and there is no injective mapping $i:\{1,\ldots,l\} \rightarrow \{1,\ldots,k\}$, such that

(1) $l > 1,\ l < k,$
(2) $T \& (L_{i(1)} \vee \ldots \vee L_{a(l)})\theta \Longrightarrow \exists B\theta$
(3) $T \& B\tau \Longrightarrow \exists(L_{i(1)} \vee \ldots \vee L_{a(l)})\tau$ for a variablefree substitution τ with neutral constants.

4.2 An effective subsystem

The concept of complete separation is recursively enumerable, but in general not recursive. Furthermore for arbitrary theories the search for proofs is very costly even if a solution exists. Therefore we are looking for an effective and efficient subsystem. The essential step to such subsystem is of course the restriction of the background knowledge, such that a suitable deduction procedure

exists. Analysing lemma 1 we see, that if we drop the provable completeness, we can restrict background knowledge to Horn clauses and goal clauses without reintroducing the problems discussed in paragraph 2.

First the concept of complete separation has to be substituted by the weaker concept of separation.

Definition:

A set of conditions $L_i=(\{B_{i1},\ldots,B_{imi}\},\alpha_i)$ is a separation of the condition $L=(\{A_1,\ldots,A_n\},\alpha)$ in respect to the theory T and the set F of case data for α, iff

(1) for all unclassified case data $F_u=(M,\beta,\theta)$ with α subset of β and for all models I of T:
(1.1) if L covers F_u in I, then there is at most one i, such that L_i covers F_u in and
(1.2) for all i: L covers F_u in I, if L_i covers F_u in I and
(2) for all models I of T and for all unclassified case data $F_u \in F$ there is an i, such that:
L covers F_u in I ==> L_i covers F_u in I.

Obviously the two separation concepts differ in respect to their completeness. While a separation covers all case data of a given set F, a complete separation covers all possible cased data. If the set M contains all possible case data, then a separation of M is a complete separation. For the induction purpose this weakening of the completeness condition is acceptable.

The following lemma shows the correctness of this intuitive reasoning.

Lemma 3:

Let $L=(\{A_1,\ldots,A_n\},\alpha)$ be a condition and $L_i=(\{B_{i1},\ldots,B_{imi}\},\alpha)$ be a set of conditions. With exception of α let all variables be different. Let T_H be a set of Horn clauses and T_G a set of goal clauses. Let $T=T_H \cup T_G$ and let $\theta(\tau)$ be a variablefree substitution of $L_i(L)$ with constants occuring neither in L_i nor L nor T. Let F be a set of unclassified case data for α. Then L_i is a separation of L in respect to the background knowledge T and the case data F, iff

(1) $T_H \& L_i\theta \Longrightarrow \exists L\theta$ for all i,
(2) for all $i \neq j$, there is a $C \in T_G$, such that $T_H \& \exists(L_i \wedge L_j)\theta \& C$ inconsistent and
(3) for all $F_u=(M,\delta,\sigma) \in F$ there is an i, such that $T_H \& M\sigma \& L\sigma\tau \Longrightarrow \exists L_i\sigma\tau$.

Definition:

Let (L,α) be a condition and let $(\{L_i\},\alpha)$ and $(\{K_j\},\alpha)$ be separations of L in respect to the background knowledge T and the set M of case data for α.

The separation $(\{L_i\},\alpha)$ is called finer than the separation $(\{K_j\},\alpha)$, iff there is a k, such that after suitable renumbering: $\{L_1,\ldots,L_k\}$ is a separation of K_1 in respect to T and M.

Lemma 4:

Let $L=(\{A_1,\ldots,A_n\},\alpha)$ be a condition. Let θ be a variablefree substitution of L_i with constants occuring in neither L_i nor L nor T. A separation $L_i=(\{B_{i1},\ldots,B_{im_i}\},\alpha)$ $(i=1,\ldots,k)$ of L in respect to the background knowledge T and the set M of case data for α is a coarsest, proper separation of L in respect to T and M, iff there is no conjunction of atoms B, containing the variables α, not having in common any variable with L or L_i and there is no injective mapping $f:\{1,\ldots,l\} \rightarrow \{1,\ldots,k\}$, such that

(1) $1 < l < k$ and
(2) $T_H \;\&\; L_{i(j)}\theta \Rightarrow \exists B\theta$ for all $j=1,\ldots,l$
(3) $T_H \;\&\; B\tau \Rightarrow \exists L\tau$ for some suitable variablefree substitution
(4) for all $i \notin f\{1,\ldots,l\}$: there is a $C \in T_B$, such that
$T_H \;\&\; C \;\&\; (B \wedge L_i)$ is inconsistent.

4.3 An Efficient Subsystem

Under quite natural conditions (finite set of case data, no infinite proofs) the concept of separation is effective (i.e. decidable). But a computation of separations is though feasible still quite inefficient. We discuss a procedure to determine separations later. Now we will do the last step in our logical analysis.

The reason for inefficiency of the computation of separations is the proof of consistency (i.e. disjointness of a separation). We therefore introduce a new weaker concept: refinement.

Definition:

A set of conditions $L_i=(\{B_{i1},\ldots,B_{im_i}\},\alpha_i)$ is a refinement of the condition $L=(\{A_1,\ldots,A_n\},\alpha)$ in respect to the theory T and the set F of case data for α, iff

(1) for all unclassified case data $F_u=(M,\beta,\theta)$ with α subset of β, for all models I of T and for all i:
L covers F_u in I, if L_i covers F_u in I and
(2) for all models I of T and for all unclassified case data $F_u \in F$ there is an i, such that:
L covers F_u in I $\Rightarrow$ L_i covers F_u in I.

Obviously refinements may contain conditions L_i, L_j $(i=/=j)$ such that there are unclassified case data F_u covered by L_i and L_j in respect to some model I of T. The problem here is that one might have the theory T={A<-B, A<-C, B<-C, C<-B}. (Note that this theory allows infinite proofs.) Then B, C is a refinement of A although the sets of case data covered by B and C are identical. In this degenerated case the resultant set of rules is redundant since it contains two rules IF ... AND B AND ... THEN ... and IF ... AND C AND ... THEN ... with the same meaning. But since B and C cover the same set of case data, the refinement B,C will not yield any gain of information and therefore will be discarded by the hill climbing search.

In general overlapping refinements may yield some gain of information and may therefore be used, if there is no more informative refinement. This does not create any logical problem, but demands a different interpretation of the decision tree and the rules generated from it. One example can travel down more than one branch respectively more than one rule can be applied. The conclusion may therefore not be one but can be one out of several possible

solutions.

Again we state the correctness of the idea and the relation of refinements and generalised expansion rules.

Lemma 5:

Let $L=(\{A_1,\ldots,A_n\},\alpha)$ be a condition and $L_i=(\{B_{i1},\ldots,B_{im_i}\},\alpha)$ be a set of conditions. With exception of α let all variables be different. Let T_H be a set of Horn clauses. Let $\theta(\tau)$ be a variablefree substitution of $L_i(L)$ with constants occuring neither in L_i nor L nor T. Let F be a set of unclassified case data for α. Then L_i is a refinement of L in respect to the background knowledge T and the case data F, iff

(1) T_H & $L_i\theta$ ==> $\exists L\theta$ for all i and
(2) for all $F_u=(M,\delta,\sigma) \in F$ there is an i, such that
T_H & $M\sigma$ & $L\sigma\tau$ ==> $\exists L_i\sigma\tau$.

Definition:

Let (L,α) be a condition and let $(\{L_i\},\alpha)$ and $(\{K_j\},\alpha)$ be refinements of L in respect to the background knowledge T and the set M of case data for α.

The refinement $(\{L_i\},\alpha)$ is called finer than the refinement $(\{K_j\},\alpha)$, iff there is a k, such that after suitable renumbering: $\{L_1,\ldots,L_k\}$ is a refinement of K_1 in respect to T and M.

Lemma 6:

Let $L=(\{A_1,\ldots,A_n\},\alpha)$ be a condition. Let θ be a variablefree substitution of L_i with constants occuring in neither L_i nor L nor T. A refinement $L_i=(\{B_{i1},\ldots,B_{im_i}\},\alpha)$ (i=1,...,k) of L in respect to the background knowledge T and the set M of case data for α is a coarsest, proper refinement of L in respect to T and M, iff there is no conjunction of atoms B, containing the variables α, not having in common any variable with L or L_i and there is no injective mapping $i:\{1,\ldots,l\} \rightarrow \{1,\ldots,k\}$, such that

(1) $1 < l < k$ and
(2) T_H & $L_{i(j)}\theta$ => $\exists B\theta$ for all j=1,...,l and
(3) T_H & $B\tau$ ==> $\exists L\tau$ for some suitable variablefree substitution.

Lemma 6 shows that a coarsest refinement of L is generated by resolving one literal B in L with all possible clauses of T_H. Let $B_1:-C_1$, ..., $B_k:-C_k$ be the set of all clauses of T_H, such that B_i can be unified with B having σ_i as most general unifier. Then a coarsest refinement of L is $LC_1\sigma_1$, ..., $LC_k\sigma_k$, where LC_i is derived from L by replacing B by C_i. Other coarsest refinements can be generated by the introduction of new descriptors. Since L & f_i ==> L holds for an arbitrary set of formulae f_i, we will later restrict the generation of expansion rules based on this kind of refinements.

5. A brief description of NINA 2

The basic procedure of NINA 2 is like that of ID3 as given in the generalised description of paragraph 3 using generalised expansion rules. Paragraph 4 demonstrates how to determine the generalised expansion rules according to some theory representing background knowledge.

First there are expansion rules to introduce new descriptors not yet

contained in the premise, i.e. rules like P,empty -> $L_1,\ldots,L_n$. These expansion rules can be considered representing the theory Ø, i.e. the logical background knowledge of pure predicate logic.

To represent any non-logical background knowledge, there are further expansion rules based on a non-empty theory T. If we demand that the case data are separated into disjoint classes at each expansion step, then we have to make use of Lemma 3 and 4. We have to determine the separations. To each separation $L_1,\ldots,L_k$ of L there is an expansion rule P,L --> $L_1,\ldots,L_k$, which has to be computed from the background knowledge. A rough outline of this computation is as follows:

Let L be a condition. Apply resolution to one of the literals of L to generate all possible goal trees for L using the Horn clause theory T_H. (This is the same as the backward chaining of a pure prolog program.) Let the goal trees be represented by the relation rel1 on the set of goals. Take all thus generated goals G as axioms and prove all constraints of T_G. Any such proof states a symmetric relation rel2 between two goals. Now compute all total subgraphs of (G,rel2). A total subgraph gr is called finer than gr' if all of its goals are goals of gr' or are below one of gr' according to rel1. According to lemma 4 the set of all total graphs, which are maximal in respect to the relation "finer" are coarsest separations of L. For those we generate the generalised expansion rules.

To have an idea of the computational costs of generating expansion rules, we can compare the computation to running a prolog program. A complete search of the proofs of L has to be accomplished (like :- L,fail.). This defines the relation rel1 on goals. Then the possible proofs of G_i have to be generated (like :-member(<- G, T_G),G,fail.). The last step introduces the relation rel2. Than we have to find all total subgraphs of rel2.

Finally all total subgraphs of (G,rel2) which are not maximal in respect to the reflexive and transitive closure of rel1 are discarded. (This are the total subgraphs whose nodes are subnodes of another total subgraph in respect to the relation <, which is defined by

```
G < G.
G1 < G2:-
   rel1(G1,G),
   G < G2.
```

If we contend ourselves with refinements, then the computational effort is much lower. Lemma 6 shows that a coarsest refinement of L is generated by resolving one literal B in L with all possible clauses of T_H. Let B_1:-C_1, ..., B_k:-C_k be the set of all clauses of T_H, such that B_i can be unified with B having σ_i as most general unifier. Then a coarsest refinement of L is $LC_1\sigma_1$, ..., $LC_k\sigma_k$, where LC_i is derived from L by replacing B by C_i.

Other coarsest refinements can be generated by the introduction of new descriptors. Since L & f_i ==> L holds for an arbitrary set of formulae f_i, we have to restrict the generation of expansion rules P,empty --> $f_1,\ldots,f_k$ based on this kind of refinements. This is done by explicitely listing such refinements and imposing a partial ordering on them. The details depend on the language used. Let us assume, that we use an attribute language and that in some way a partial ordering called "coarser" is defined on the set of attributes used to build decision trees. Only if all attributes "coarser" than a except a itself are contained in L, then we allow the expansion rule L,empty --> $a=v_1$, ..., $a=v_k$.

rel1 defined by goal trees build by applying resolution to clauses of the theory T induces a relation rel3 on the set of attributes. rel3 holds between two attributes a and b, if a is contained in a goal g1, b in goal g2 and rel1(g1,g2) hold. If rel3 is a subset of the relation "coarser", then the allowed expansion rules represent coarsest refinements of L. If this is not the case the algorithm might produce an overspecialised decision tree. If we insist on using coarsest refinements only, we have to construct the theory T in a hierarchical mannor. First we introduce the partial ordering "coarser". Then we prohibit clauses, which when chained together lead from an attribute a to an attribute b with a "coarser" than b.

At present NINA 2 operates in refinement mode only. A preprocessor generates expansion rules from a Horn clause theory. A generalised ID3 algorithm using generalised expansion rules is run to build a decision tree. This algorithm employs prolog as a theorem prover on the theory T in order to separate the case data according to the expansion step. A post processor generates rules from the tree.

6. Conclusions

Based on a logical analysis an efficient induction algorithm can be described, able to process background knowledge as discussed in paragraph 2. The procedure defined this way is a generalisation of ID3. In ID3 all predicates are unary. The only argument is instantiated by a value specific to the case datum. Usually the predicate is called attribute. It is assumed that for each object an attribute has exactly one value.

The equality species=cat in the attribute logic of ID3 corresponds to the expression species(cat) in the language of first order predicate logic that we used here. If we want to use separations we have to make explicite that species=cat and species=dog are contradictory. In predicate logic this assumption can be expressed by the goal clause

```
<-species(cat),species(dog).
```

The example demonstrates, that any ID3 application can be emulated by NINA 2 in refinement and separation mode. To use separation mode a set of goal clauses T_G can be defined to explicitely express the disjointness of equations like species=cat and species=dog.

Some of the problems address here have also been address in the INSTIL project. In [6], [7] and [8] a system called NEDDIE is described, which is a generalisation of ID3. It employs an object oriented representation language which allows to express class hierarchies. Class hierarchies are a special case of background knowledge as understood in this paper. We may for example represent cats and dogs as subclasses of mammmals. This can be expressed by Horn clauses `mammal(X) <- cat(X), mammal(X) <- dog(X).`

To see that class hierarchies represent only a special case of background knowledge we discuss the use of a recursive definition as background knowledge.

```
Let T={size in [A,B] :- size in [A,C], C is (A+B)/2,
       size in [A,B] :- size in [C,B], C is (A+B)/2 ).
```

From T we can generate the expansion rule:

```
P,size in [A,B]  -->  size in [A,C], C is (A+B)/2,
                      size in [C,B], C is (A+B)/2
```

This expansion rule can be used recursively to divide an interval several times until the intervals are small enough to differentiate the case data or until other attributes yield a greater gain of information.

7. Literature

[1] Buntine, W.: Generalised Subsumption and its Applications to Induction and Redundancy, Proceedings of the 7th European Conference on Artificial Intelligence, Brighton 1986

[2] Quinlan, J.R.: Learning Efficient Classification Procedures and their Application to Chess End Games, in: Machine Learning, An Artificial Intelligence Approach, Michalski, R.S., Carbonell, J.G., Mitchell, T.M., Tioga Publishing Company 1983

[3] Lloyd, J.W.: Foundations of Logic Programming, Springer Verlag 1984

[4] Monk, J.D.: Mathematical Logic, Springer Verlag 1976

[5] Shapiro, E.Y.: Algorithnmic Programm Debugging, MIT Press 1983

[6] Blythe, J.S.: Constraining Search in a Hierarchical Discriminative Learning System, in: Proceedings of the European Conference on Artificial Intelligence, 1988

[7] Manago, M.: Object oriented generalisation: a tool for improving knowledge based systems, in: Proceedings of the International Meeting on Advances in Learning, Les Arcs 1986

[8] Blythe, J.S. et al.: Knowledge Acquisition by Machine Learning: the INSTIL Project, ESPRIT Conference 1988

[9] Bundy, A., Silver, B., Plummer, D.: An analytic comparison of some rule learning programs, Expert System 1983

This work has been part of the LERNER project. The LERNER project was a cooperation between TU-Berlin, Stollmann GmbH and Nixdorf Computer AG. It has partly been supported by the German Ministry of Research and Technology under grant ITW 8501.

EKAW-89

Architectures for Next-Generation Knowledge Acquisition Systems

EKAW-89 Tutorial Notes: Interactive Knowledge Acquisition Tools

Paris, France, July, 1989

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

At a series of knowledge acquisition workshops, papers on knowledge acquisition tools and techniques discussed *manual* methods (for instance, protocol analysis), *automated* tools (traditional machine learning techniques), *interactive* tools, or combinations of these (Boose, 1989). This tutorial concentrates on the interactive tools. Interactive knowledge acquisition tools can be associated with knowledge-based application problems and problem-solving methods. This descriptive approach provides a framework for analyzing and comparing tools and techniques, and focuses the task of building knowledge-based systems on the knowledge acquisition process.

Interactive Knowledge Acquisition Tools

Musen proposed that knowledge acquisition tools could be associated with specific problems or specific problem-solving methods (Musen, 1987). In a related manner, we have worked to classify tools with problems *and* problem-solving methods, since most problems are strongly linked to certain types of problem-solving methods. Consequently, certain types of domain knowledge and possibly control knowledge should be acquired to build the corresponding knowledge-based system. This idea was discussed at the First AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop held in Banff, Canada, in November, 1986 (Gaines and Boose, 1989). Builders of interactive knowledge acquisition tools were asked to try and classify their work and the work of others in terms of these relationships. Figure 1 (Boose, 1989) shows a possible mapping of such relationships at a high level in Clancey's problem classification hierarchy and a problem-solving method classification hierarchy (Clancey, 1986). Lower levels in the problem hierarchy would be sub-problems (i.e., trouble shooting and symptom analysis would be found under diagnosis), and the leaves of the problem hierarchy would be specific application problems to be solved.

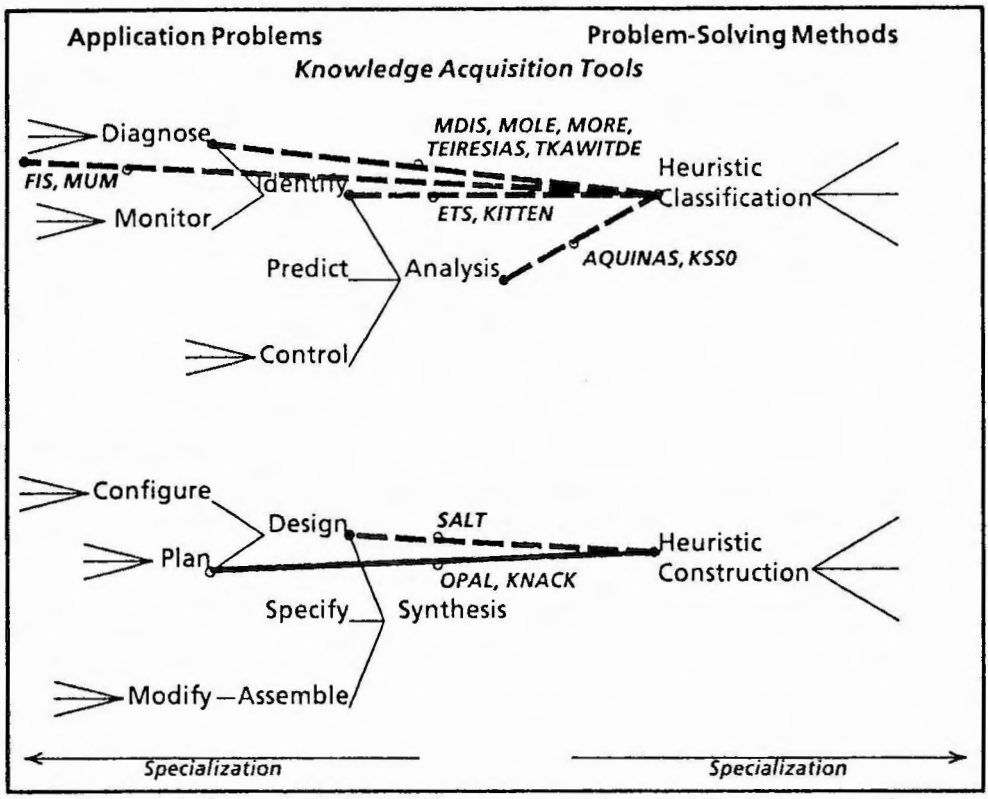


Figure 1. Knowledge acquisition tools may be associated with relationships between application problems and problem-solving methods. Representative tools shown above include are listed below (tools are discussed in more detail in Boose, 1989).

Representative tools shown in Figure 1 include:

AQUINAS - *elicit and model information using a knowledge acquisition workbench including hierarchically-structured repertory grid-based interviewing and testing and other methods* (Boose and Bradshaw, 1987b), (Kitto and Boose, 1987), (Boose and Bradshaw, 1987a), (Boose, 1988), (Bradshaw and Boose, 1989), (Kitto and Boose, 1988), (Shema and Boose, 1988), (Boose, Bradshaw, and Shema, 1988)

ETS - *interview experts using repertory grid-based methods and test the knowledge* (Boose, 1984, 1985, 1986a, b)

FIS - *tie knowledge acquisition closely to the fault diagnosis domain* (De Jong, 1987)

KITTEN - *interview experts using repertory grid-based methods* (Shaw and Gaines, 1987), (Shaw and Woodward, 1988)

KNACK - *elicit and use knowledge about evaluation report generation* (Klinker, Bentolila, Genetet, Grimes, and McDermott, 1987), (Klinker, Genetet, and McDermott, 1988)

KSS0 - *elicit knowledge with a repertory grid-based interviewing tool including text analysis, behavior induction, and psychological scaling techniques* (Gaines, 1987a, 1987b, 1988), (Gaines and Sharp, 1987), (Shaw and Gaines, 1987), (Shaw, 1988)

MDIS - *experts are interviewed to describe mechanisms in a top-down structured manner for diagnostic problems* (Antonelli, 1983)

MOLE - *exploit information about how problems are solved to elicit scarce diagnostic knowledge and use feedback to fine tune the knowledge* (Eshelman, Ehret, McDermott, and Tan, 1987), (Eshelman, 1988)

MORE - *exploit information about how problems are solved to elicit extensive diagnostic knowledge* (Kahn, Nowlan, McDermott, 1985a,b)

MUM - *evidential combination knowledge and control knowledge are elicited for medical problems* (Gruber and Cohen, 1987)

OPAL - *tie knowledge acquisition closely to the cancer treatment domain* (Musen, Fagan, Combs, and Shortliffe, 1987)

SALT - *elicit and deliver knowledge for constructive constraint satisfaction tasks* (Marcus, McDermott, and Wang, 1985), (Marcus, 1987), (Stout, Caplain, Marcus, and McDermott, 1988)

TEIRESIAS - *model existing knowledge to monitor refinements and help debug consultations* (Davis and Lenat, 1982)

TKAW/TDE - *exploit information about how problems are solved to elicit trouble-shooting knowledge* (Kahn, Breaux, Joeseph, and DeKlerk, 1987)

Associations between problem domains and problem-solving methods help define the depth and breadth of current knowledge acquisition development. For example, Boeing's work using AQUINAS has led them to try and build a broad link (multiple integrated tool sets) between a general application problem class (analysis problems) and a powerful problem-solving method (heuristic classification). Other successful work has led developers to tightly couple knowledge acquisition tools to a domain problem (for example, FIS, STUDENT, OPAL).

Associations in the problem-method framework where no tools exist can point out promising areas for new projects. For example, can special types of knowledge acquisition tools be associated with debugging problems and heuristic classification, or with planning and new specializations of heuristic construction?

A more detailed analysis of this classification, descriptions of tools, and development strategies associated with them are contained in (Boose, 1989). This study also lists manual techniques and automated (machine learning) techniques presented at the workshops, and provides a catalog of tools and methods.

Knowledge Acquisition Tool Dimensions

AQUINAS was used prior to one of the workshops to classify the tools and develop tool dimensions.

Knowledge in AQUINAS is represented, in part, in networks of repertory grids. Objects appear along one axis of a grid and dimensions or traits appear along the other axis. AQUINAS helps the expert develop, analyze, refine, and test knowledge. A grid showing twenty-six interactive knowledge acquisition tools and associated analyses is shown in (Boose, 1989).

AQUINAS elicited the following set of interactive knowledge acquisition tool descriptors (again, these are discussed in more detail in Boose, 1989):

Application task dimensions

- Level of generality (domain dependence) - How domain-dependent is the tool?
- Analysis / synthesis - What broad categories of application tasks can the tool address?
- Specific problem - Has the tool been built for a specific problem? If so, what is the problem?
- Application statistics (number and size of applications) - How many applications have been built with the tool? How diverse are they? How large are they? How much of the finished system did the tool help build?

Knowledge acquisition techniques and methods - What general knowledge acquisition techniques are employed?

- Psychology-Based and Interviewing Methods
    - o Automated / Mixed-initiative Interviewing - *the tool interviews the expert*

- o Protocol Analysis (Case Walk-Through / Eidetic Reduction / Observation / Process-Tracing) - *record and analyze transcripts from experts thinking aloud during tasks*
- o Psychological Scaling (including multidimensional scaling) - *use scaling techniques to help structure knowledge*
- o Repertory Grids / PCP - *use personal construct psychology and related methods to elicit and analyze knowledge*

- Task / Method / Performance Exploitation
  - o Domain Exploitation (Single Application) - *rely heavily on the domain for knowledge acquisition guidance*
  - o Problem-Solving Method Exploitation - *use information about the problem-solving method to guide knowledge acquisition*
  - o Performance System *(direct link or embedded) - generate knowledge that may be directly tested and used*
- Modeling
  - o Decision Analysis - *perform probabilistic inference and planning using influence diagrams*
  - o Modeling *(deep models, causal models, cognitive models, conceptual models, mediating representations, task-level models) - use or generate models of the domain, possibly independent of a tool or a specific application*
  - o Consistency Analysis - *analyze knowledge for consistency*
  - o Physical Model Simulation - *use basic laws to derive physical models through simulation*
- Multiple Experts
  - o Delphi - *gather information from people independently*
  - o Multiple Source - *elicit and analyze knowledge from multiple sources separately and combine for use*
- Other Sources of Knowledge
  - o Textual Analysis / Natural Language Analysis - *generate knowledge directly by analyzing text*

Modeling dimensions

- Deep modeling / shallow modeling - Are "deep" models or "causal" models elicited?
- Multiple / single methods for handling uncertainty - What techniques, if any, are used to model uncertainty?

Representations

- Expertise representation method ("intermediate" representation) (cognitive maps, concepts, correlations, decision tables, frames, goal structures, hierarchies, operators, probability distributions, relations, repertory grids, rules, scripts, tables)
- Knowledge types (causal knowledge, classes, conceptual structures, constraints, control, covering, example cases, explanations, facts, goals, judgments, justifications, preferences, procedures, relations, spatial, strategic, temporal, terminology, uncertainties)

Features

- High-level techniques / low-level techniques - How sophisticated are the techniques used?
- Learning component (automatic, interactive, none) - If there is a learning component in the tool, how powerful is it? Is it automatic or interactive?
- Multiple features / few features - How many techniques are integrated in a single framework? How well do multiple techniques support each other?
- Multiple knowledge sources support - Is there specific support for eliciting, analyzing, or delivering knowledge from multiple experts or other sources?
- Multiple knowledge views / few knowledge views - How many ways are there to look at elicited knowledge? What, if any, knowledge transformation techniques are employed?

System use

- Automated tool / semi-automated tool / manual technique - How much of the technique is implemented as a computer program? How "smart" is the tool? Is effective tool use dependent on the user, or does the tool offer semi-automated or automated assistance?
- Efficiency of use; speed of use - How hard is the tool to use? How efficient is knowledge elicitation and modeling? How well are the techniques implemented?
- Implementation stage (planned, in progress, implemented,tested, in use, past use)
- Intended users (end-users of expert system, decision makers, experts, knowledge engineers, AI programmers needed) - Who are the targeted users of the tool?
- Life cycle support (one-shot use to complete cycle support) - How much of the knowledge engineering and system delivery life cycle does the tool support?
- System in use / system not in use - Is the tool currently in use? Was the tool previously in use? Will the tool be in use in the future?
- Training needed - How much training is needed to use the tool? Can experts use the tool directly?
- Validation, verification, maintenance - Does the tool offer support for testing and maintenance?

AQUINAS performed several analyses of the knowledge. For example, an implication analysis produced by AQUINAS showed logical entailments between different dimensions. A similarity analysis among dimensions showed, for example, that EFFICIENT.AND.FAST.TO.USE was closely coupled to LITTLE.TRAINING.NEEDED. A similarity analysis among tools showed, for example, high similarity between ETS, KITTEN, and PLANET, and low similarity between FIS and KSS0 (similarity scores were produced for each pair of dimensions and each pair of tools).

AQUINAS also produced several "scatter tables" showing clusters of tools plotted on successive pairs of dimensions. A simplified version of one of these tables, domain independence vs. task class, is shown in Figure 2. The table shows concentrations of interactive knowledge acquisition tools for diagnostic tasks, but the few knowledge acquisition tools that exist for synthesis problems are domain-dependent. Other example tables showed that generally, it is easier to automate tools that are more domain-dependent; that, generally, automated knowledge acquisition tools are easier to learn how to use; and that knowledge acquisition tools that support more of the knowledge engineering life cycle tend to be more domain-dependent.

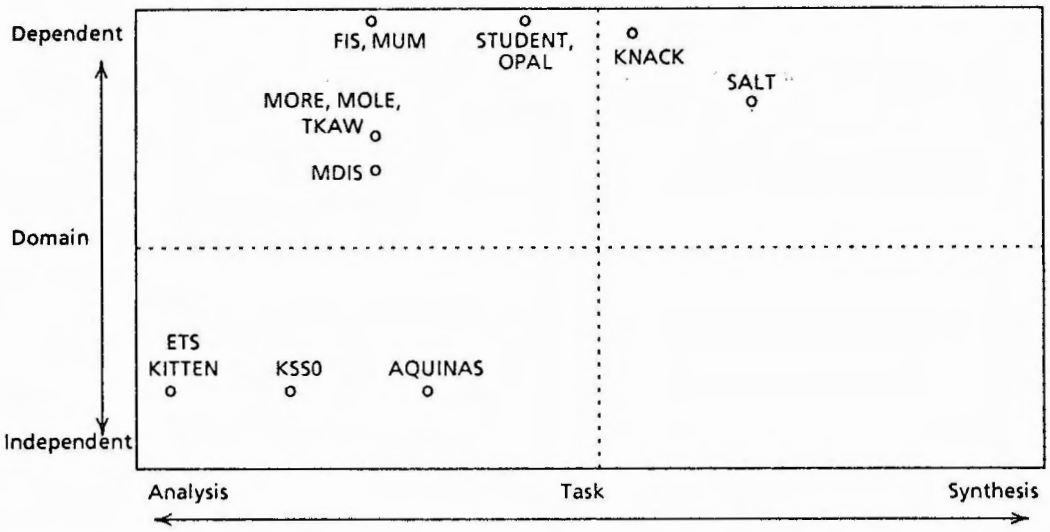


Figure 2. AQUINAS interactive knowledge acquisition tool scatter table for domain independence vs. task class. The table shows concentrations of interactive knowledge acquisition tools for diagnostic tasks, but the few knowledge acquisition tools that exist for synthesis problems are domain-dependent.

Other patterns in the tools are apparent. For instance, some tools try to draw power using strong specific domain knowledge (FIS, OPAL, MUM); other tools try to address a broader range of problems at the expense of built-in domain-specific problem-solving power (ETS, KITTEN, AQUINAS). The few tools that address synthesis problems are domain dependent. Most developers seem to be interested in applying their tools to more domain independent and/or harder tasks.

Success in implementing interactive knowledge acquisition tools for particular problems seems to follow success at building large numbers of knowledge-based systems in that domain. Once many systems exist, patterns become apparent that may be exploited by specialized knowledge-based system shells, and in turn, by knowledge acquisition tools. For example, many diagnostic knowledge-based systems exist, specialized diagnostic shells are appearing, and we see a strong concentration of knowledge acquisition tools in this area. As more specific synthesis knowledge-based systems are built, and useful specialized shells emerge for these kind of problems, we should expect to see more knowledge acquisition tools for synthesis problems.

Knowledge Acquisition Workshops and Related Publications

We are attempting to make the knowledge acquisition workshop materials as widely available as possible. The following section details the availability of publications from each workshop and from a new Academic Press knowledge acquisition journal. If you have related questions contact Brian Gaines (gaines@calgary.cdn) or John Boose (john@atc.boeing.com).

First AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW86), Banff, November 1986

Preprints distributed to attendees only.
Revised and updated papers published in the *International Journal of Man-Machine Studies*, January, February, April, August and September 1987 special issues.
Papers plus editorial material and index collected in two books in the Knowledge-Based Systems Series: Gaines, B.R. & Boose, J.H. (Eds) Vol. 1:

Knowledge Acquisition for Knowledge-Based Systems. London: Academic Press, 1988. Boose, J.H. & Gaines, B.R. (Eds) Vol. 2:Knowledge Acquisition Tools for Expert Systems. London: Academic Press, 1988.

First European Knowledge Acquisition for Knowledge-Based Systems Workshop (EKAW87), Reading, UK, September 1987

Proceedings available as:
Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems. Send sterling money order or draft for 39.00 pounds payable to University of Reading to: Prof. T.R.Addis, Department of Computer Science, University of Reading, Whiteknights, PO Box 220, Reading RG6 2AX, UK.

Second AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW87), Banff, October 1987

Preprints distributed to attendees only.
Revised and updated papers being published in the *International Journal of Man-Machine Studies*, 1988 regular issues (August, September, October, November, December, and others). Papers plus editorial material and index will be collected in book form for the Knowledge-Based Systems Series, together with other knowledge acquisition papers from IJMMS. These will be available in spring, 1989: Gaines, B.R. & Boose, J.H. (Eds) Vol. 3: Machine Learning and Uncertain Reasoning in Knowledge-Based Systems. London: Academic Press, 1989. Boose, J.H. & Gaines, B.R. (Eds) Vol. 4: Knowledge Acquisition Tools and Methods for Knowledge-Based Systems. London: Academic Press, 1989.

Second European Knowledge Acquisition for Knowledge-Based Systems Workshop (EKAW88), Bonn, West Germany, June 1988

Proceedings available as:
Boose, J.H., Gaines, B.R., and Linster, M. (Eds), *Proceedings of the European Workshop on Knowledge Acquisition for Knowledge-Based Systems (EKAW88).* Send order to (the GMD will invoice you for DM68.00 plus postage): Marc Linster, Institut fur Angewandte Informationstechnik der Gesellschaft fr Mathematik und Datenverarbeitung mbH, Schloss Birlingoven, Postfach 1240, D-5205 Sankt Augustin 1, West Germany.

Third AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW88), Banff, November 1988

Proceedings / preprints available as:
Boose, J.H. & Gaines, B.R. (Eds), *Proceedings of the 3rd Knowledge Acquisition for Knowledge-Based Systems Workshop.* Send money order, draft, or check drawn on US or Canadian bank for US$65.00 or CDN$85.00 to: SRDG Publications, Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N 1N4. Revised and updated papers will be published in the *International Journal of Man-Machine Studies*, 1989 regular issues, and possibly in the proposed Academic Press journal, *Knowledge Acquisition for Knowledge-Based Systems.* Papers plus editorial material and index will be collected in book form, together with other knowledge acquisition and related papers from IJMMS and KAKBS in 1990.

Information about proceedings and publications from the workshops below will be available later; get in touch with the listed contact for future information.

Third European Knowledge Acquisition for Knowledge-Based Systems Workshop (EKAW89), Paris, France, July 1989

Contact Jean-Gabriel Ganascia, LAFORIA, Univ. Pierre et Marie Curie, Tour 45-46, 4 Place Jussieu, 75230 Paris Cedex 05; (32)-1-69 41 66 26.

Fourth AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW89), Banff, October 1-6 1989

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

Fourth European Knowledge Acquisition for Knowledge-Based Systems Workshop (EKAW90), The Netherlands, Summer, 1990

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

Fifth AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW90), Fall, 1990

Information about proceedings and publications from this workshop will be available in the future. Contact Brian Gaines (address above).

First Japan Knowledge Acquisition for Knowledge-Based Systems Workshop (JKAW90), Fall, 1990

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

Journal: ***Knowledge Acquisition: An International Journal of Knowledge Acquisition for Knowledge-Based Systems***, Academic Press, Editors: Brian R.Gaines & John H.Boose, first scheduled for publication in June, 1989.

Aims: *Knowledge Acquisition* aims to provide those developing knowledge-based systems with access to state-of-the-art research on tools, techniques, methodologies and foundations for knowledge acquisition from experts, texts and other sources of human knowledge. The emphasis of this journal is on the support of human knowledge processes through advanced knowledge-based systems, and particularly on the elicitation and modeling of knowledge in those systems derived from more conventional sources. The emphasis is not on artificial intelligence but on the extension of natural intelligence through knowledge-based systems.

Coverage

Elicitation/modeling of expertise: systems that obtain and model knowledge from experts.

Elicitation/modeling of expertise: manual knowledge acquisition methods and techniques.

Apprenticeship, explanation-based, and other learning systems; integration of such systems with other knowledge acquisition techniques.

Integration of knowledge from multiple experts or other sources; issues in collaborative knowledge-based systems.

Issues in cognition and expertise that affect the knowledge acquisition process.

Extracting and modeling of knowledge from text.

Integration of knowledge acquisition techniques within a single system; integration of knowledge acquisition systems with other systems (hypermedia, database management systems, simulators, spreadsheets).

Knowledge acquisition methodology and training.

Validation of knowledge acquisition techniques; the role of knowledge acquisition techniques in validating knowledge-based systems.

References

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

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, p. 27-33.

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

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

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

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

Boose, J. H. (1989). A Survey of Knowledge Acquisition Techniques and Tools, *Knowledge Acquisition: An International Journal of Knowledge Acquisition for Knowledge-Based Systems*, in press, Vol. 1, No. 1.

Boose, J. H., and Bradshaw, J. M. (1987a). Expertise Transfer and Complex Problems: Using Aquinas as a Knowledge Acquisition Workbench for Expert Systems, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 1, *International Journal of Man-Machine Studies*, Vol. 26, No.1, p.3-28; also in Boose, J. H., and Gaines, B. R. (eds), Knowledge-Based Systems Vol. 2: Knowledge Acquisition Tools for Expert Systems, New York: Academic Press, 1988, p. 39-64.

Boose, J. H., and Bradshaw, J. M. (1987b). AQUINAS: A Knowledge Acquisition Workbench for Building Knowledge-Based Systems, *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, September, p. A6.1-6.

Boose, J. H., Bradshaw, J. M., Shema, D. B. (1988). Recent Progress in Aquinas; A Knowledge Acquisition Workbench, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, p. 2.1-15.

Bradshaw, J. (1988). Shared Causal Knowledge as a Basis for Communication Between Expert and Knowledge Acquisition System, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, p. 12.1-6.

Bradshaw, J. M. (1988). Strategies for Selecting and Interviewing Experts, Boeing Computer Services Technical Report, in preparation.

Bradshaw, J. M., and Boose, J. H. (1989). Decision Analytic Techniques for Knowledge Acquisition: Combining Situation and Preference Models Using Aquinas, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, in press.

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

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

De Jong, K. (1987). Knowledge Acquisition for Fault Isolation Expert Systems, special issue on the 1st AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, *International Journal of Man-Machine Studies*, Vol. 27, No. 2.

Eshelman, L, Ehret, D., McDermott, J., and Tan, M. (1987). MOLE: A Tenacious Knowledge Acquisition Tool, special issue on the 1st AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 1, *International Journal of Man-Machine Studies*, Vol. 26, No.1, p. 41-54; also in Boose, J. H., and Gaines, B. R. (eds), Knowledge-Based Systems Vol. 2: Knowledge Acquisition Tools for Expert Systems, New York: Academic Press, 1988, p. 95-108.

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*, in press.

Gaines, B. R. (1987a). An Overview of Knowledge Acquisition and Transfer, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 3, *International Journal of Man-Machine Studies*, Vol. 26, No.4, p. 453-472; also in Gaines, B. R., and Boose, J. H. (eds), Knowledge-Based Systems Vol. 1: Knowledge Acquisition for Knowledge-Based Systems, New York: Academic Press, 1988, p. 3-22.

Gaines, B. R. (1987b). Knowledge Acquisition for Expert Systems, *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, September, p. A3.1-4.

Gaines, B. R. (1988a). Advanced Expert System Support Environments, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, in press.

Gaines, B. R. (1988b). Second Generation Knowledge Acquisition Systems, *Proceedings of the Second European Knowledge Acquisition Workshop (EKAW-88)*, Bonn, June, p. 17.1-14.

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

Gaines, B. R., Sharp, M. (1987). A Knowledge Acquisition Extension to Notecards, *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, September, p. C1.1-7.

Gruber, T. R., Cohen, P. R. (1987). Design for Acquisition: Principles of Knowledge System Design to Facilitate Knowledge Acquisition, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 2, *International Journal of Man-Machine Studies*, Vol. 26, No.2, p. 143-160; also in Boose, J. H., and Gaines, B. R. (eds), Knowledge-Based Systems Vol. 2: Knowledge Acquisition Tools for Expert Systems, New York: Academic Press, 1988, p. 131-148.

Kahn, G., Nowlan, S., McDermott, J. (1985a). Strategies for Knowledge Acquisition, IEEE Transactions of Pattern Analysis and Machine Intelligence, PAMI-7 (3), p. 511-522.

Kahn, G., Nowlan, S., and McDermott, J. (1985b). MORE: An Intelligent Knowledge Acquisition Tool, in the *Proceedings of the Ninth Joint Conference on Artificial Intelligence*, Los Angeles, CA, August, p. 581-584.

Kahn, G. S., Breaux, E. H., Joeseph, R. L., and DeKlerk, P. (1987). An Intelligent Mixed-Initiative Workbench for Knowledge Acquisition, special issue on the 1st AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, *International Journal of Man-Machine Studies*, Vol. 27, No. 2, p. 167-180; also in Boose, J. H., and Gaines, B. R. (eds), Knowledge-Based Systems Vol. 2: Knowledge Acquisition Tools for Expert Systems, New York: Academic Press, 1988, p. 161-174.

Kitto, C. M., and Boose, J. H. (1987). Heuristics for Expertise Transfer: The Automatic Management of Complex Knowledge Acquisition Dialogs, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 2, *International Journal of Man-Machine Studies*, Vol. 26, No. 2, p. 183-202; also in Boose, J. H., and Gaines, B. R. (eds), Knowledge-Based Systems Vol. 2: Knowledge Acquisition Tools for Expert Systems, New York: Academic Press, 1988, p. 175-194.

Kitto, C. M., and Boose, J. H. (1988). Selecting Knowledge Acquisition Tools and Strategies Based on Application Characteristics, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, in press.

Klinker, G., Bentolila, J., Genetet, S., Grimes, M., and McDermott, J. (1987). KNACK: Report-Driven Knowledge Acquisition, special issue on the 1st AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 1, *International Journal of Man-Machine Studies*, Vol. 26, No.1, p. 65-80; also in Boose, J. H., and Gaines, B. R. (eds), Knowledge-Based Systems Vol. 2: Knowledge Acquisition Tools for Expert Systems, New York: Academic Press, 1988, p. 195-210.

Klinker, G., Genetet, S., McDermott, J. (1988). Knowledge Acquisition for Evaluation Systems, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, in press.

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

Marcus, S. (1987). Taking Backtracking with a Grain of SALT, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 3, *International Journal of Man-Machine Studies*, Vol. 26, No. 4, p. 383-398; also in Boose, J. H., and Gaines, B. R. (eds), Knowledge-Based Systems Vol. 2: Knowledge Acquisition Tools for Expert Systems, New York: Academic Press, 1988, p. 211-226.

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, special issue on the 1st AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 1, *International Journal of Man-Machine Studies*, Vol. 26, No.1, p. 105-121; also in Boose, J. H., and Gaines, B. R. (eds), Knowledge-Based Systems Vol. 2: Knowledge Acquisition Tools for Expert Systems, New York: Academic Press, 1988, p. 257-274.

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, p. 5.1-15.

Shaw, M. L. G., and 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, p. 251-280.

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, p. 329-350.

Shema, D. B., and Boose, J. H. (1988). Refining Problem-Solving Knowledge in Repertory Grids Using a Consultation Mechanism, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, p.447-460.

Stout, J., Caplain, G., Marcus, S., McDermott, J (1988). Toward Automating Recognition of Differing Problem-Solving Demands, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, in press.

THE KNOWLEDGE DICTIONARY: STORING DIFFERENT KNOWLEDGE REPRESENTATIONS

Bob Jansen* & Paul Compton†

ABSTRACT

This paper describes interim results of research in the area of the software engineering of knowledge based systems. It discusses several problems identified in the knowledge base work by the research group, especially in the area of design, documentation and maintenance. One of the major areas of concern is identified as the lack of integration of the various 'standard' knowledge representation formalisms. The research aims to test the application of conventional software engineering methodologies to the software engineering of knowledge based systems. The methods employed include the entity-relationship analysis of various knowledge representations, and the documentation of these representations in a data dictionary augmented to enable the definition and description of knowledge in addition to information and data. This environment is called the Knowledge Dictionary. The results are discussed in the context of this environment which is used as a test environment for the software engineering of an expert system, Garvan-ES1. The paper describes conceptual models of some of the standard knowledge representations, how the Knowledge Dictionary may be used as an aid to integrate some of these representations, the benefits in doing so, and discusses the relationship that exists between knowledge representation and knowledge acquisition.

KEYWORDS
data dictionary, knowledge representation, production rules, semantic nets, frames, ripple down rules, knowledge dictionary, database software, knowledge engineering, relational calculus, knowledge acquisition

ELECTRONIC MAIL ADDRESSES
ACSnet: jansen@ditsyda.oz CSNET: jansen@ditsyda.oz
JANET: ditsyda.oz!jansen@ukc ARPA: jansen% ditsyda.oz@ seismo.css.gov
UUCP: {enea, hplabs, mcvax, prlb2, seismo, ubcvision, ukc}! munnari !ditsyda.oz!jansen
AUSPAC: jansen@au.csiro.ditsyda

THIS PAPER IS AN AMENDED COPY OF THE PAPER PRESENTED AT THE FIFTH CONFERENCE ON THE APPLICATIONS OF EXPERT SYSTEMS , HELD IN SYDNEY AUSTRALIA MAY 1989.

* CSIRO Division of Information Technology,PO Box 1599, North Ryde, NSW, 2043,Australia
† Garvan Institute of Medical Research, St. Vincents Hospital, Victoria St. Darlinghurst, NSW,Australia

1. INTRODUCTION.

In recent times, the field of knowledge based systems has seen an increase in the number of methods of representing knowledge. This increase is no doubt due to the complexity of knowledge and the perception that in tackling a new problem, existing representations may be only partially suitable. This then raises the question of whether to adopt an existing representation, possibly compromising the richness inherent in the knowledge domain, or to develop a new representation suitable to capture the domain. As with conventional system's design, the choice of either option is a compromise. However, the choice of the right representation is crucial in ensuring the robustness and usefulness of the software product, and usually leads to the translation of difficult problems into simpler ones (Winston, 1984).

Various 'standard' knowledge representations exist, namely *production rules* (Davis & King,1984), *frames* (Minsky, 1974), *semantic nets* (Winston, 1984), *conceptual structures* (Sowa, 1984), *Prospector like inference networks* (Lister, 1987), and *ripple down rules* (Compton & Jansen, 1988). The common element between all of these representations is suitably summed up by Winston, "All representations must provide some way to denote objects and to describe the relations that hold among them" (Winston, 1984, page 253).

With the introduction of knowledge based systems into the commercial arena, problems arise regarding the access by the inference engine to data already stored in traditional databases. There is much work proceeding regarding solutions to this problem, (Al-Zobaidie & Grimson, 1984, Delcambre, 1988, Jansen & Compton, 1988a, Kaiser, 1988, Zaniolo, 1986) but as yet there appears to be no all encompassing solution. In addition, there are now several conferences devoted almost solely to the integration of knowledge based systems and databases (for example the Expert Database Systems workshop) showing the importance attached to this integration problem with the introduction of AI technology into the commercial computer area. The required integration introduces a major issue in the area of design and software engineering; how can a complex system, having both conventional and knowledge based components, be software engineered? There are as yet no design tools commercially available to aid the designer in the overall design process.

The work of maintaining GARVAN ES1 (Compton & Jansen, 1988) has led us to realize that knowledge is more complex than is perhaps envisaged, and may need computing solutions incorporating aspects of both holistic and reductionist strategies (Compton & Jansen, 1989). There appear to be philosophical aspects to the knowledge acquisition phase that should be taken into account to explain the problems that are being reported (eg. Shaw, 1988 on the problems of multiple experts) which need to be re-evaluated. This topic is not of major concern in this paper, but is discussed in more detail in Compton & Jansen, 1989. However, it appears that knowledge representation and knowledge acquisition go hand in hand as part of the analysis process, and not sequentially, that is, knowledge acquisition is not primarily a process of getting the expert to remember the knowledge but to acquire and represent the knowledge in a suitable integrated knowledge structure.

This paper describes preliminary results from on-going work that attempts to tackle some of these issues, to provide an environment where expert systems are maintainable for the long term, and can be integrated into large knowledge based systems. The work is centred around the use of existing software engineering techniques in the area of knowledge based systems, and developing models of software products incorporating both conventional and knowledge based components. This paper focuses on four knowledge representation formalisms, the *production rule, semantic nets, frames* and *ripple down rules*. Other knowledge representations will be the subject of future papers as the work proceeds. The environment in which the work is described is a development called the *Knowledge Dictionary* (Jansen & Compton, 1988b).

2. THE KNOWLEDGE DICTIONARY

Over the last two years, the CSIRO Division of Information Technology and the Garvan Institute of Medical research have been involved in a collaborative project aimed at the design and maintenance of expert systems, using as the target, GARVAN ES1 (Horn *et al*, 1985), an expert system for the clinical interpretation of laboratory reports. This expert system has grown more difficult to maintain over the four years of routine production use, due mainly to the refinement of the knowledge in the domain rather than an expansion of the domain (Compton & Jansen, 1988).

In parallel with this project, the CSIRO Division of Information Technology has been redeveloping SIRATAC (Hearn *et al*, 1986), a cotton pest management decision support expert system, utilizing modern software engineering practises in the redevelopment.

A major development from these two projects is a tool called the *Knowledge Dictionary* (Jansen & Compton, 1988b). This tool was derived from techniques forming part of conventional software engineering, in particular the application of data dictionary technology to the area of knowledge based systems. To date, the Knowledge Dictionary may be used to record knowledge in the form of production rules, using the relational data model (Codd, 1970) supporting the underlying representation. This representation may be manipulated into conventional production rule of the form IF...THEN.... It should be noted that although the user interface representation is the production rule, the knowledge is not stored in production rule format, but is decomposed into constituent components and stored in relational tables.

Among the advantages of representing knowledge in the Knowledge Dictionary are:-

- the knowledge may be browsed using as a starting point any concept in the knowledge domain. For example, starting at a particular rule, the user can browse to other rules, facts, and rule actions related to the starting rule in any way; use of similar facts in the premise, reaching the same conclusions etc. (See Jansen & Compton (1988)c for more details).

- the knowledge is automatically documented and cross referenced with respect to other concepts in the domain. The documentation of the knowledge and knowledge base proceeds *in situ* with the knowledge acquisition, and not as an afterthought once the knowledge base has been built.

- the knowledge may be processed in various ways using the full power of relational calculus, thus opening up to the knowledge engineer and the expert, new and novel ways of manipulating the stored knowledge. One such process is inferencing. In fact the Knowledge Dictionary has an associated inference engine that accomplishes forward chaining inference using SQL type data manipulation rather than *resolution* (see Jansen & Compton, 1988c, Delcambre & Etheredge, 1988).

- access to the knowledge is independent of any computer programming language, thus allowing easier access to the knowledge by unskilled, in a computer sense, end user experts.

- the maintenance of the knowledge is aided by storing the knowledge in a normalized form (Howe, 1986), where each concept is fully defined and named **once only**. For example, assume knowledge acquisition discovered that expert A has a concept 'tsh high', representing the condition that the tsh level in a blood sample is above a certain threshold. Another expert has the identical concept, but labeled 'tsh is above normal'. From the view of the knowledge, this is un-normalized, in that there are two representations of the same concept and maintenance would require two changes to be effected, one to each representation. The knowledge engineer may recognize this condition, and either force one expert to use the other expert's label, or alternatively, store the definition of the concept once, and map each expert's label onto this node. In this way, the expert system has the concept in a normalized form (ie. defined once and a single label), whilst the experts are able to maintain their un-normalized form and are shown the knowledge using their own terminology. This maintains the experts' familiarity with the stored knowledge, stressed in Hayes-Roth *et al* (1983) as vital for the success of building expert systems, whilst incorporating the advantages of normalization in the maintenance process.

- as with some conventional data dictionary systems (eg. the ICL 2900 Data Dictionary System), the knowledge is related to the concepts in the business or user world, and all relationships are explicit. Thus the effects of any change in either the user world or the computer world, representing the implementation of the user world, may be ascertained prior to effecting the change.

To date the Knowledge Dictionary has been implemented in both Prolog and Hypercard on a Macintosh computer, running on the Mac Plus, Mac SE, and Mac II models, as well as in the

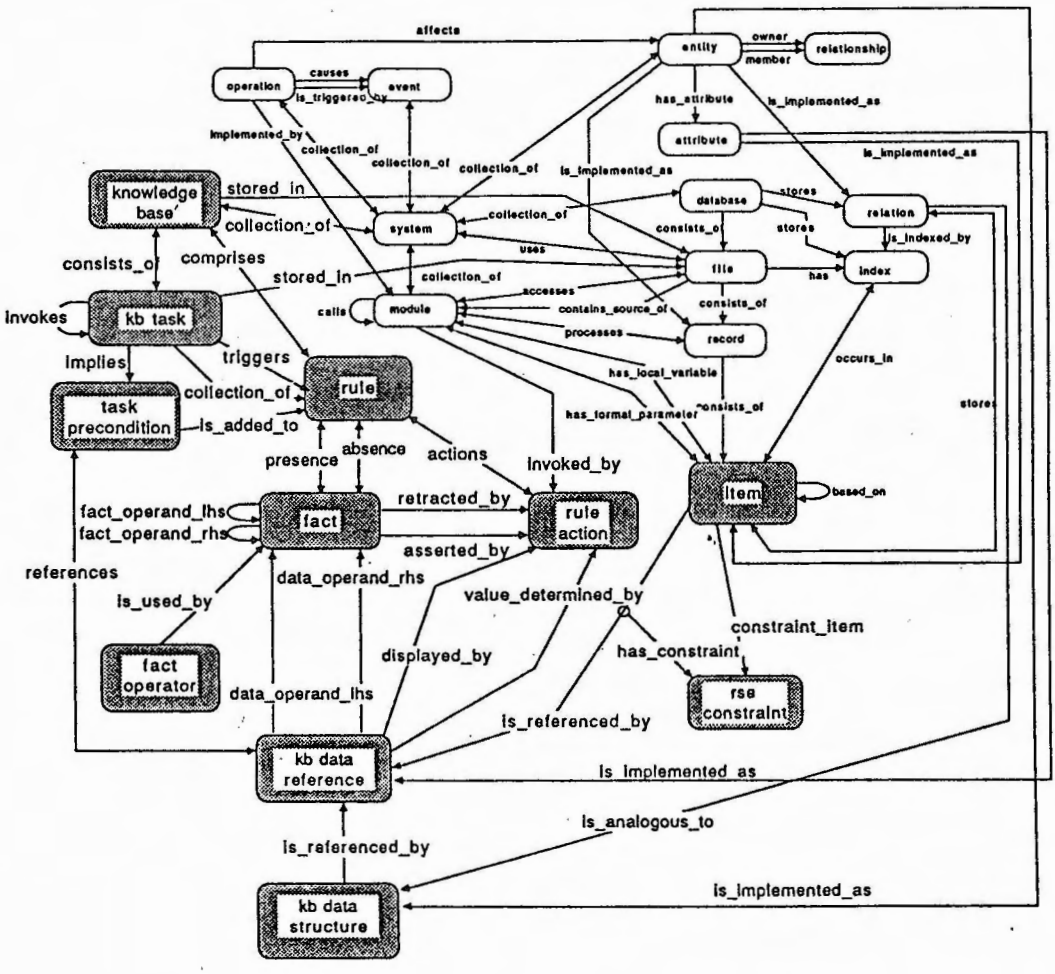


Figure 1. The Knowledge Dictionary Conceptual Model

Digital Equipment Corporation's relational database package RDB and the fourth generation software, RALLY, running on a MicroVax computer.

3. CAPTURING KNOWLEDGE REPRESENTATIONS IN THE KNOWLEDGE DICTIONARY

As stated above, the Knowledge Dictionary currently enables the capturing of knowledge in the form of production rules, and the manipulation of the stored knowledge in various ways. Obviously, the other knowledge representation formalisms must be able to be expressed as well in order for the Knowledge Dictionary to be useful in a wide variety of domains.

Work has been progressing in this area as described below.

3.1. Production Rules/The Existing Data Model.

As stated previously, the Knowledge Dictionary uses the Relational Data Model as the underlying storage representation for the concepts captured in the domain.

The conceptual data model is shown in figure 1[1]. This figure is drawn using the Entity-Relationship model formalism. It should be noted that this model is incomplete in respect to the concepts of the conventional computer system and is not the major concern here[2]. However, it does allow the capturing of the business information in the form of Entity-Relationship models (Chen, 1976) and Operation-Event models and shows the objects used in their implementation in the computer system. The shaded entities on this diagram are those entities added to the conventional conceptual model which allow the capturing of production rules in their constituent atomic form.

Conceptually, production rules are stored as follows. Each rule is decomposed into its constituent components, namely a *rule*, as set of *facts* and a set of *rule actions*. The constituent components are then stored in a table as a set of relationships between the *rule* and each *fact*, and the *rule* and each *rule action*. The complete rule can be built up by retrieving each tuple from the table and manipulating the names into the required formalism. This is demonstrated in figure 2 (see Jansen & Compton, 1988c for more details).

Rule 44.

IF tsh > 7
and tsh < 15
THEN tsh is high

Object Name	Relationship Name	Object Name
.	.	.
.	.	.
.	.	.
Rule 44	presence	fact - tsh > 7
Rule 44	presence	fact - tsh < 15
Rule 44	actions	assert fact - tsh is high
.	.	.
.	.	.
.	.	.

Figure 2. This shows the conceptual translation of a production rule into a table. The relationship names are those as shown on figure 1 between rule components.

[1] Note that in this ER diagram, the arrows point from the owner object to the owning object, or the generic to the more specific.
[2] Note that this is the subject of current ANSI work as detailed in Dolk *et al* 87 & Dolk 88.

3.2 Semantic Nets.

As described in Winston (1984), semantic nets represent objects, called *nodes*, and relationships between the objects, called *arcs* or *links*. Semantic nets must be differentiated from ordinary nets, in that semantic information, or knowledge, must be imparted when the net is viewed. This is demonstrated in figure 3.

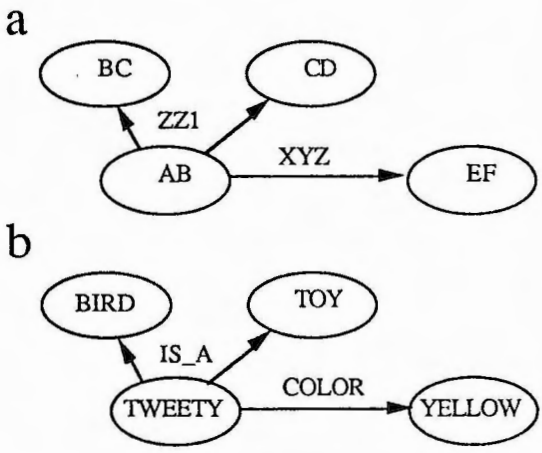


Figure 3. Two net diagrams, an ordinary net(a) and a semantic net(b). Note that the ordinary net imparts no semantic information, but may impart structural information, whereas the semantic net provides semantic knowledge about BIRDS, TOYS, and a bird/toy instance. TWEETY
Note that in semantic nets, the arrows point from the specific to the generic node.

As the concepts to be captured are objects and relationships, in a similar configuration to an Entity-Relationship model, the Knowledge Dictionary can be used to capture this diagram, and hence the knowledge, by augmenting figure 1 to include the concepts as shown in the Entity-Relationship model, figure 4.

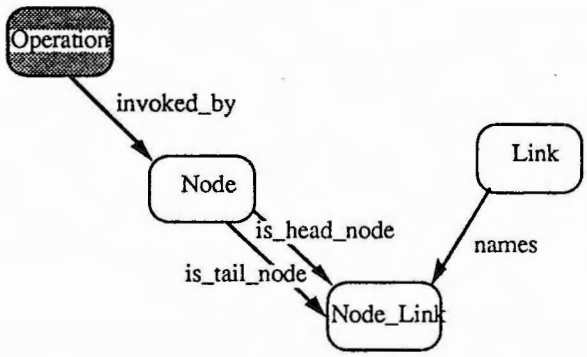


Figure 4. Concepts added to Conceptual Model to allow the capturing of semantic nets. The figure shows the relationship between the semantic net and the shaded 'operation' entity, an existing entity on the conceptual model, figure 1. We have introduced a Node_Link entity to represent the fact that a node may have multiple links with the same name, a situation not allowed in ER models. Two nodes may not be linked by multiple links of the same name however. Note this diagram is in Entity-Relationship form.

Figure 4 shows one importance difference between our semantic net representation, and that espoused by Winston (Winston, 1984). This difference is important, in that it recognizes that the semantic net diagram should be in a normalized form. In figure 5a, the semantic link 'weight (if needed)' is not normalized in that it implicitly defines two concepts within the one link, namely a *weight* for a block, and the existence of the *weight calculation procedure* if the weight is needed. In our estimation, these two concepts should be explicitly defined, as shown in figure 5b, where we draw a node for the existence of a 'weight' for the block, and a separate node, attached to the weight node, representing the existence of the 'weight calculation procedure'. This difference is important when it comes to the representation of the semantic node as a frame, as in the Winston

form, it is possible for a slot to have many facets, whereas in our representation, their is a single facet to a slot, although the facet may have more than one value.

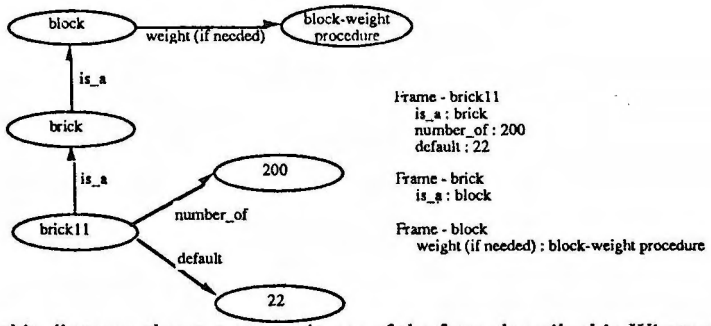


Figure 5a - this diagram shows a semantic net of the form described in Winston 84. It details a non-normalized link, 'weight (if needed)' between the nodes 'block' and 'block-weight procedure'. The equivalent frame representation of the diagram is also shown.

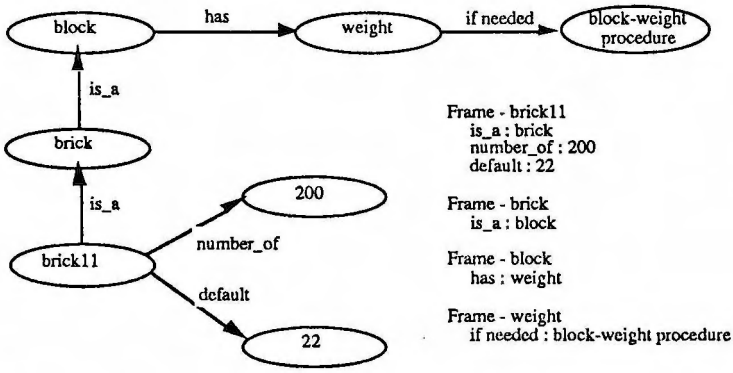


Figure 5b - this shows the 'normalized' representation of the 'weight (if needed)' link from figure 5a. Tis representation is preferred, in that it explicitly recognizes that there are two concepts, namely a weight and a procedure to calculate it if required.The equivalent frame representation of the diagram is also shown.

The semantic net in figure 3b would conceptually be represented in table entries as shown in table 1.

Node Entries	Node Link Entries	Link Entries
Tweety	Tweety:is_a:Bird	is_a
Bird	Tweety:is_a:Toy	colour
Toy	Tweety:colour:Yellow	
Yellow		

Table 1. This table shows the conceptual representation of the simple semantic net in figure 3b.

The above model can be shown capable of representing the mechanisms of *inheritance*, *demons*, *defaults*, and *perspectives* , described in Winston (1984), as detailed below.

Inheritance is the term used to describe the facility of passing down some or all of the properties of the more generic node to a more specific node. This is analogous to the database concept of subtyping. The inheritance mechanisms is not an explicit part of the diagram itself, but a manipulation of the knowledge on the diagram, represented by the links between nodes. Thus inheritance is not visible either in the Knowledge Dictionary representation. In figure 3, the node

TWEETY could inherit the properties of the nodes BIRD and TOY. This scenario makes sense of the statements that 'TWEETY is_a BIRD', and 'TWEETY is_a TOY'.

The *demon* mechanism is a mechanism for automatically performing an action in the event of a specific situation. For example, the weight calculation procedure, shown in figures 5a and 5b would be implemented as a demon if the weight of the block is required. The demon mechanism is represented and documented using the *invoked_by* relationship between an *operation* and a *node* entity.

The *perspective* mechanism is a method of recognizing that the semantics of an object may vary depending on the current point of view of the observer. This allows us for example, to describe a person as happy when on holidays, but unhappy when at work. Thus the properties associated with the object may vary depending on the current perspective. This mechanism is represented by a semantic link between the perspective nodes and parent node, in the same way as other links.

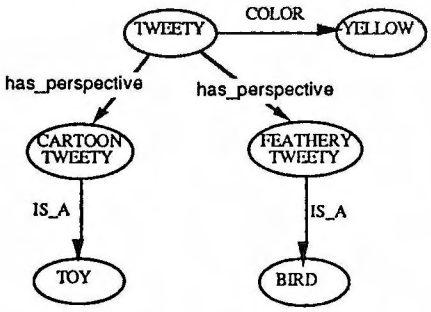


Figure 6 - Diagram showing how the perspective mechanism is handled in the semantic net representation by the 'has_perspective' links. These links are treated as any other link by the model shown in figure 4.

3.3. Frames.

Winston (1984) states that a *frame* is a collection of semantic net nodes and slots (or *link* in figure 4) that together describe a stereotyped object, act, or event. This statement implies that, given that a semantic net may be represented in the Knowledge Dictionary as shown in figure 4, then a frame can be considered a transformation of the information from the semantic net representation. This is analogous to the case of the IF...THEN... representation of a production rule, which is a transformation of the information stored within the Knowledge Dictionary, as shown by the shaded entities on figure 1.

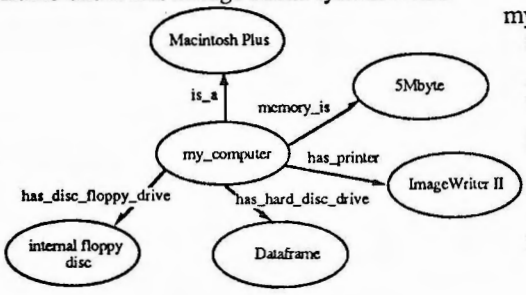
Cuadrado & Cuadrado (1986) introduce frames in a simple and understandable way shown in figure 7 using Backus-Naur notation.

Frame	::=	frame_name slot ...
Slot	::=	slot_name facet...
Facet	::=	(facet_name value) ...
	::=	demon_name
Frame_name	::=	character string
Slot_name	::=	character
Facet_name	::=	character
Demon_name	::=	module_name
Value	::=	number
	::=	character

Figure 7. Backus-Naur representation of a frame as described in Cuadrado & Cuadrado 86

Comparison of this representation with the semantic net representation shown in figure 4, leads to the conclusion that the frame representation can be generated from the semantic net representation, recognizing the *one-to-one* correspondence of 'node's to 'frame's, and the *one-to-many* correspondence between 'slot's and 'node_link's.

Thus to present a frame based interface for a user, the Knowledge Dictionary would have to be front ended with appropriate graphic based translation code, a task independent of the major software engineering of the system. To allow frame based processing by a commercially available frame based knowledge based system would




```
my_computer
  memory_is
        value : 5Mbyte
  has_printer
        value : ImageWriter II
  has_hard_disc_drive
        value : Dataframe
  has_floppy_disc_drive
        value : internal floppy disc
  is_a
        value : Macintosh Plus
```

Figure 8. A diagram showing on the left a semantic net representation and on the right a frame based representation for equivalent information about 'my computer'.

require a generator to translate the semantic net representation within the Knowledge Dictionary to the required data structure in an external file[1].

Figure 8 shows the semantic net representation represented in the Knowledge Dictionary and a generated frame based representation. Note that the semantic net representation is a conceptual representation of the actual underlying storage structure as implemented in the current Knowledge Dictionary. See Jansen & Compton (1988)b and Jansen (1989) for a details.

The similarity of frames and semantic nets is a useful feature when considering the generation of frame structures, especially if one classifies the semantic net as the user view of the knowledge, and the frame as the computer implementation of the knowledge. This classification allows the computer implementation of the knowledge to be hidden from the users or experts, but visible to the knowledge engineer. However, if the knowledge is represented naturally in a frame representation, then the expert should not be forced into manually translating to a semantic net representation first. If the semantic net representation is required, then this should be capturable automatically from the frame descriptions. On the other hand, if the expert's representation is in the semantic net form, then automatic tools should be available to generate the appropriate frame structure in the dictionary from the semantic net description. Thus the Knowledge Dictionary must be capable of

> capturing semantic net descriptions directly, coupled with a 'semantic net to frame' translator facility.
>
> and the capturing of frames directly, coupled with a 'frame to semantic net' translator facility.

This may be seen as analogous to the relationship between *entities* and *records* in conventional data structures. A record description is able to be generated from an entity description, but the dictionary still has a *record* entity to describe record structures separately. This is done to aid the maintenance task, where it may be important to know, what modules access a record type. This complies with the basic dictionary philosophy of representing the user world and it's implementation in the computer world. Thus an *entity* is an object in the user world, whereas a *record* is an implementation in the computer world. Similarly, a *node* is a representation in the user world, whereas a *frame* is a representation in the computer world. Thus extra entities must be added to the conceptual diagram, figure 4. The amended entity diagram is shown in figure 9.

[1]It is planned that the Knowledge Dictionary will have a generator capable of generating any defined representation and its own frame based knowledge based system. The translation tables, or grammar of the required representation would be stored as meta knowledge within the Knowledge Dictionary. Currently the generation facility needs to be specifically programmed for each representation.

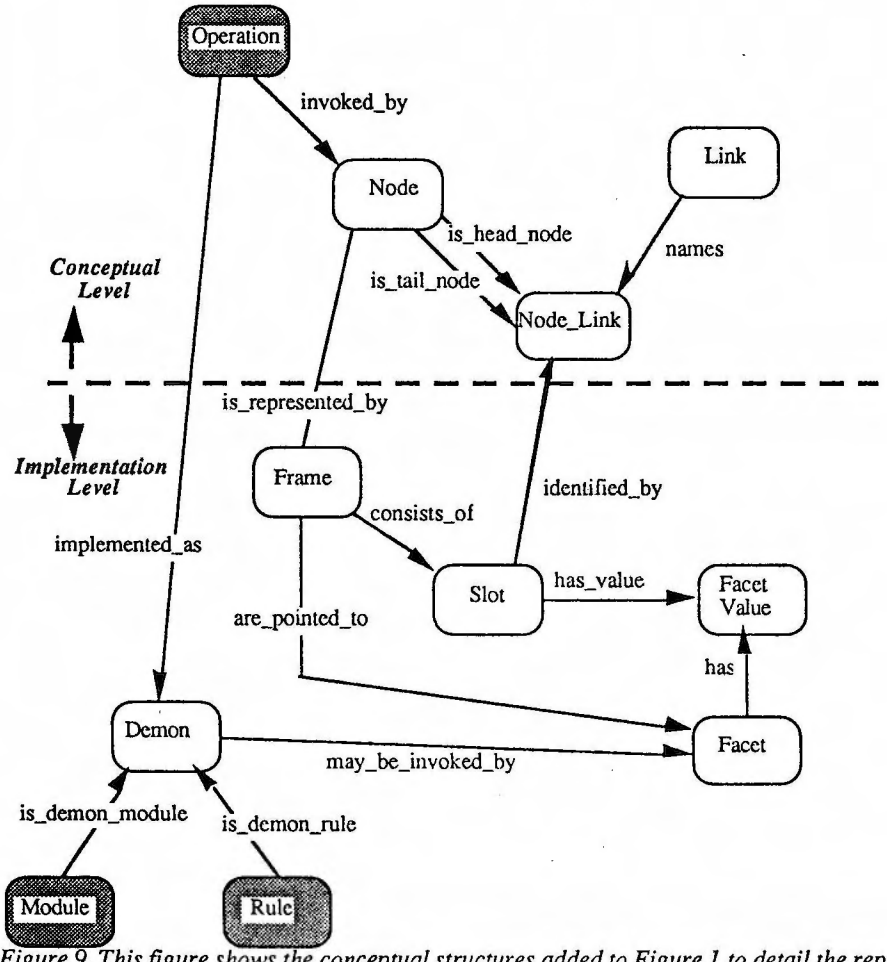


Figure 9. This figure shows the conceptual structures added to Figure 1 to detail the representation of a semantic net and its implementation as a frame in the computer world. Note the relationships to existing entities 'operation', 'module', and 'rule'. There is a 'mutual exclusivity' constraint between the 'are_pointed_to' and 'may_be_invoked_by' relationships relating to a 'facet'. A 'facet' may either point to a frame, or invoke a demon, but not both.

A major benefit in using the Knowledge Dictionary can be seen in this case, as the demon facility can now be linked to a 'rule' in a rule based system, thus invoking for example, a forward chaining operation starting from the related rule. Thus, a slot in a frame can cause the execution of part of a production rule expert system, instead of, as in most existing frame based systems, a module of executable code. This is a significant advance in the integration of two knowledge base representations.

3.4. Ripple-Down Rules

The concept of *Ripple-Down Rules* (Compton & Jansen, 1988) was developed in an attempt to capture the context of a particular rule. This came about as a result of the realization that diagnostic rule-based systems undergo maintenance because the expert verbalizes a rule within the context of a currently failing case. The expert system then attempts to apply this new rule across the complete domain whenever possible, whereas we surmise that the new rule should only be used if the current area of investigation is the rule that initially failed and the current profile matches that given by the expert as the reason for having a 'fix-up' rule.

Ripple_down rules were so named because instead of fixing an existing rule, the knowledge engineer appends a 'fix up' rule to the knowledge base. This fix up rule contains as one condition, that the last rule to fire was the particular existing rule that is the subject of the maintenance. This condition gives the context for the maintenance operation.

For a more detailed description of the Ripple_Down Rule mechanism, see Compton & Jansen (1988) or Compton & Jansen (1989) (paper this workshop).

The implementation of the ripple down rule mechanism in the Knowledge Dictionary is very simple. Ripple-down rules contain a condition LAST_FIRED(..), where the number in brackets indicates the rule number of the last rule that must have fired. This gives the rule the context, that unless the last rule that fired has the rule number identical to that in the LAST_FIRED (..) condition, the new rule can not fire. Thus the LAST_FIRED(..) condition is treated as an extra fact in the premise of the rule.

The simple solution is to define an extra relationship called *last_fired* from a *rule* object type to another *rule* object type. This is shown in figure 10.

The only other change required is to the generate function, generating IF...THEN production rules for use by the Garvan-ES1 expert system. This function was enhanced to test for the *last_fired* relationship, and if it existed, to include a LAST_FIRED(..) condition in the rule's premise.

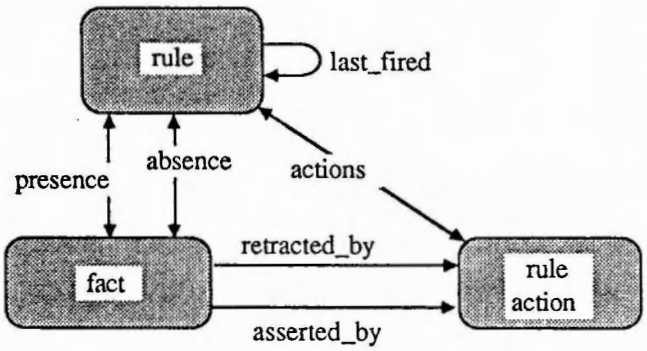


Figure 10. A subset of the conceptual model of the Knowledge Dictionary, shown in figure 1, for a production rule utilizing the Ripple-Down rule approach. The addition is the LAST_FIRED relationship between rule object types.

4. KNOWLEDGE ACQUISITION AND KNOWLEDGE REPRESENTATION

The above discussions have focussed on the area of different knowledge representations, and suggesting underlying data structures enabling them to be implemented in the Knowledge Dictionary environment. However, it appears that knowledge representation plays an important part in the knowledge acquisition process as well.

Compton & Jansen (1989) discusses the problems of knowledge acquisition, and their relation to philosophical questions on the nature of knowledge. These views, in conjunction with the problems being discovered in the knowledge acquisition process, highlight that knowledge should be seen as a complex resource whose structure and meaning depend on the current context of the observer. Shaw (1988) points out that the knowledge acquisition process involving multiple experts is complicated by the fact that experts use of the same labels for different concepts or different labels for the same concepts. We suggest that this infers that they utilize different structures for the representation of their knowledge, and in particular we maintain that the expert's knowledge structure at any time depends on acquisition and representation context. Thus, given the complexity of knowledge, how does a knowledge engineer begin and guide the knowledge acquisition process to a successful conclusion?

From experience gained with the redevelopment of SIRATAC and GARVAN ES1, it appears that the visualization of the knowledge in suitable representations (Rappaport & Gaines, 1988) is a suitable vehicle for this guidance process, in a similar way that a system's analyst will

draw several versions of entity-relationship models for a domain being implemented in a database regime as part of the analysis process. These modelling formalisms only act to draw the analysist's attention to logical subsets and relationships (or focal points) in the knowledge. These focal points may subsequently be used to aid and manage the systematic extraction of the knowledge from the experts.

The production of representation schemes as part of knowledge acquisition should not be seen as a holy grail, but rather as working documents that are liable to change as the knowledge acquisition proceeds. Further knowledge will alter focii and the possible relationships established in the already captured knowledge. At any point in time, the knowledge we have is the best currently available, but its properties will change with the introduction of new knowledge.

In order for representation to guide acquisition, the knowledge engineer requires flexible tools to visualize the knowledge in various levels of detail; to be able to alter the focus of the current view of the knowledge; and to be able to simply change between representations to suit the current focus. To accomplish these tasks, models of the different knowledge representation formalisms are required, supported by tools capable of performing the translation and visualization tasks. The Knowledge Dictionary is a tool aimed at supporting these types of operations.

5. SUMMARY

In this paper, we have discussed some of the problems that exist for the knowledge engineer and expert in implementing the domain knowledge for a knowledge based system. These problems included documentation of the expert system, variety of knowledge representation formalisms and the lack of one suitable formalism. We presented the results of work being carried out in the area of the software engineering of knowledge based systems, especially the application of data dictionary concepts the knowledge base area. We described an environment called the *Knowledge Dictionary* which consists of an augmented data dictionary able to represent, document and cross reference knowledge in the form of production rules. Having entered the knowledge in the *Knowledge Dictionary,* the knowledge is accessible by various techniques not normally catered for in current expert systems, techniques analogous to those that have proved invaluable in the long time support of conventional computer systems. We then went on to demonstrate the power of the *Knowledge Dictionary* environment and the relational data model supporting the underlying representation, by showing how three other knowledge representation formalisms could be included in the *Knowledge Dictionary* environment. In doing so, the knowledge engineer could chose an appropriate representation for either the whole domain knowledge or a part of the domain. In the latter case, each representation was linked into the complete domain, so an overall picture could be obtained.

This paper ends up in a discussion regarding the relationships between knowledge acquisition and knowledge representation, and proposes that knowledge representation, in a way analogous to the drawing of ER models as part of conventional analysis, is a technique that may be used to guide and focus the acquisition process. To be successful in this task, the tools used must be capable of representing knowledge in various formalisms, and be capable of translating between them. This paper treats only four formalisms, production rules, semantic nets, frames, and ripple down rules. The modelling of other formalisms is the subject of further research. The tasks of translation between knowledge representations, and the generation of the run time knowledge base have not been treated in any great details, and although recognized as significant, present purely technical challenges.

6. ACKNOWLEDGEMENTS

The research results described in this paper are part of a collaboration project between the CSIRO Division of Information Technology and the Garvan Institute of Medical Research. We acknowledge the advice and guidance of the following people.

Bob Colomb, Neil Ashburner and Craig Lindley of the CSIRO Division of Information Technology.
Leslie Lazarus and other domain experts of the Garvan Institute of Medical Research.

7. REFERENCES

Al-Zobaidie A & Grimson J B, *Expert Systems and Database Systems: How can They Serve Each Other?*, Expert Systems, February 1987, Vol. 4. no. 1

Chen P. P, *The Entity-Relationship Model: Towards a Unified View of Data*, ACM TODS 8, 1 March 1976, pp 7-42

Codd E F, *A Relational Model for Large Shared Data Banks*, CACM, Vol. 13, No. 6, 1970.

Compton Paul, Horn Kim, Quinlan J R, Lazarus L, Ho K, *Maintaining an Expert System*, Proceedings of the Fourth Australian Conference on the Applications of Expert Systems, Sydney, 1988, pp 110-129

Compton Paul & Jansen Bob, *Knowledge in Context: A Strategy for Expert Systems Maintenance*, proceedings of AI'88 Conference, Adelaide, November 1988

Compton Paul & Jansen Bob, *A Philosophical Basis for Knowledge Acquisition*, Proceedings of the EKAW-89, Paris France, July 1989.

Cuadrado John L & Cuadrado Clara Y, *AI in Computer Vision*, Byte January 1986, pp 237-258

Davis Randall & King Jonathan J, *The Origin of Rule-Based Systems in AI, in Rule-Based Expert Systems, the MYCIN Experiments of the Stanford Heuristic Programming Project*, Bruce G. Buchanan & Edward H. Shortliffe (eds), Addison-Wesley, 1984, pp20-52

Delcambre Lois M. L. & Etheredge James N., *The Relational Production Language: A Production Language for Relational Databases*, Expert Database Systems. Proceedings of the Second International Conference on Expert Database Systems, Kerschberg (ed) April 1988

Dolk Daniel R & Kirsch II Robert A , *A Relational Information Resource Dictionary System*, Communications of the ACM, January 1987, Volume 30, Number 1

Dolk Daniel R, *Model Management and Structured Modelling: The Role of an Information Resource Dictionary System*, Communications of the ACM, Volume 31, Number 6, June 1988

Hayes-Roth Frederick , Waterman Donald A & Lenat Douglas B (editors), *Building Expert Systems*, Advanced Book Program, Addison-Wesley Publishing Company Inc. 1983

Hearn B., Brook K., Ashburner N., Colomb R., *SIRATAC A Cotton Crop Management Expert System*, Proceedings of the 1st Australian Artificial Intelligence Congress, Melbourne, 1986

Horn K A, Compton P, Lazarus L, Quinlan R, *An Expert System for the Interpretation of Thyroid Assays in a Clinical Laboratory*, The Australian Computer Journal, Volume 17, No 1, February 1985

Howe D R , *Data Analysis for Data Base design*, Edward Arnold (Publishers) ISBN 0-7131-3481-X, 1986

Jansen Bob, *Applying Software Engineering Concepts to Rule Based Expert Systems*, in AI & Software Engineering, Derek Partridge (ed), Ablex, 1989, in press

Jansen Bob & Compton Paul, *The Knowledge Dictionary: An Application of Software Engineering Techniques to the Design and Maintenance of Expert Systems*, Proceedings of the AAAI-88 Workshop on Integration of Knowledge Acquisition and Performance Systems, August 1988, Minnesota USA

Jansen Bob & Compton Paul, *The Knowledge Dictionary: A Relational Tool for the Maintenance of Expert Systems*, Proceedings of the International Conference on Fifth Generation Computing Systems (FGCS'88), Tokyo, November 1988

Jansen Bob & Compton Paul, *The Knowledge Dictionary : A Data Dictionary Approach to the Maintenance of Expert Systems*, Knowledge- Based Systems, John Gero (ed), Vol 2, No. 1, March 1989, in press

Kaiser Gail E, Barghouti Naser S, Feiler Peter H & Schwanke Robert W , *Database Support for Knowledge-Based Engineering Environments*, IEEE Expert, Summer 1988

Lister R , Asli K , Buda R, Horsfall C, Buntine W, *GOLD - an Expert System for Mineral Identification from Reflectance Spectra*, AI'87 Conference Proceedings, Sydney 1987

Minsky Marvin, *A Framework for Representing Knowledge*, Cambridge MA: MIT AI Memo #306, 1974.

Rappaport A T, Gaines B R, *Integrating Knowledge Acquisition and Performance Systems*, Proceedings of the Australian Joint Artificial Intelligence Conference (AI'88), Adelaide, 1988, pp317-336.

Shaw Mildred L G, *Validation in a Knowledge Acquisition System with Multiple Experts*, Proceedings of the International Conference on Fifth Generation Computer Systems 1988, Tokyo, 1988, pp 1259-1266

Sowa John F, *Conceptual Structures*, Addison Wesley (pub), 1984

Winston Patrick Henry, *Artificial Intelligence*, Addison-Wesley (pub), 1984.

Zaniolo Carlo, Ait-Kaci Hassan, Beech David, Cammerata Stephanie, Kerschberg Larry & Maier David, *Object Oriented Database Systems and Knowledge Systems*, Expert Database Systems Proceedings from the first International workshop, Larry Kerschberg (ed), 1986

A Methodology for Analysing Modalities of System/User Cooperation for KBS

Paul de Greef and Joost Breuker

Department of Social Science Informatics
University of Amsterdam, Herengracht 196
1016 BS Amsterdam, The Netherlands
Electronic mail: degreef@swivax.UUCP

Abstract

In this paper we discuss a methodology -MAC- which supports a process of successive refinement in analysing and defining how user and a knowledge based system (KBS) can cooperate. MAC is fully compatible with the KADS methodology [5], [6], but can also be used independently. The method starts with a highly abstract task decomposition and results in a model of cooperation, that is, a conceptual model of the prospective KBS that states which interactions should be performed when. This model of cooperation provides the constraints for interface design. In this respect MAC can be viewed as a further elaboration of Moran's [19] framework towards a cooperative view on system/user interaction. MAC not only provides a top-down framework for developing the functionality of KBS, but also defines the role of collecting empirical data which are used for *testing* assumptions in the models about roles and competences of the user. As an example of the use of MAC we present a case-study -Statcons- in the domain of advice on statistical models for empirical research.

1 Modalities of KBS

One of the reasons why the user-interface of most current KBS is far less sophisticated than that of many (direct manipulation) interfaces of conventional systems is the fact that the role of these KBS is invariably the same: performance of a single expert task. The pattern of interaction -or rather: cooperation- follows the MYCIN scenario. Problem specification and data entry are performed in a system driven way, i.e. the systems probes for specific types of data according to the present state of the process of problem solving. Problem solving and communication with the user are not separated, which invariably results in a dialogue that is not transparent to the user. From the user's point of view the dialogue is unstructured, because a current topic may have no relation with a previous one: the pragmatic rules of discourse are simply ignored.

As a back-up facility in this interaction the user may ask for "explanation", which is a justification for asking the question to the user in terms of the problem solving steps of the system -which may be intransparent as well. It should be noted that in this scenario this modality is the only one in which the user can take the initiative: in conventional systems system initiative is rather the exception.

Recapturing one could say that a the traditional advisory KBS have only three *modalities:* elicitation of data; presentation of solution or advice, and explaining the system's need for data. Modality can is the role (function) of a system in the cooperation with the user within the context of a distribution of tasks. Modality is related to the concept mode in conventional systems: mode refers to an explicit grouping of functions, which supports a structured use of a system -if these groupings coincide with (learnable) task decompositions of users. We use the term modality here because modality not only refers to task decomposition but also to task distribution.

In this paper we present a framework -or methodology- for specifying the modalities of operational KBS. MAC (Methodology for the Analysis of Cooperation) is a specification language for constructing models of cooperation, but also outlines a (partial) sequence of knowledge engineering activities. This sequence is the consequence of dependencies that exist in the "top-down" framework between the more abstract and more specific layers. This framework is to a large extent inspired by notions from Distributed AI [3], job analysis [1], the design of user interfaces, in particular [19] and [20], and discourse plan-

ning, e.g. [2] [21].

In this paper we will illustrate MAC by describing how the models of cooperation for StatCons were developed [9]. StatCons advices researchers about appropriate data collection plans and models for analysing statistical data from experiments. A prototype system -StatCons1 [11] - was developed using the KADS methodology [5], [6]. KADS was concerned with the design of the problem solver rather than the interaction or cooperation between system and user. In expert systems, in general, emphasis is placed on having a "good" problem solver and perhaps a fancy interface. Also in StatCons-1 this is the case. In testing the StatCons-1 prototype with real users it turned out that this is not enough to guarantee a smooth interaction. Many types of users were just not capable to work with the system. It seems that something is missing which is fundamental for highly interactive expert systems and we call it cooperation.

In the KADS project, which concerns commercial development of operational systems, similar problems have occurred in other domains. To amend for these problems the KADS methodology has been extended with MAC [10].

In the next section we present an overview of the framework on which the methodology is based. In the remainder of the paper we present the StatCons example.

2 The development of a cooperative system

MAC provides a framework (modeling language) to design cooperation between user and KBS. Moreover, MAC suggests methods of empirical data collection and steps in empirical testing the model of cooperation. Forms of cooperation are called "modalities".

Cooperation as it is used in MAC is based upon at least the following prerequisites. First, it assumes a task decomposition whereby the agents each perform different subtasks. This is called a task distribution. Second, the agents should know when and how their tasks fit into this task distribution. This prerequisite refers to the control aspects of cooperation. A third prerequisite is that the dependencies between these tasks have to be respected, which implies that the results of subtasks have to be transferred in time to dependent other subtasks. In general, this transfer is accomplished by communication tasks, but there also exist forms of cooperation without communication (c.f. [18]).

The control aspects of cooperation may be dynamic and adapted to the type of problem. Then the task distribution is not fixed: decompositions may change and the use of resources may have to be replanned, etc. (c.f. "multi-agent planning", [14] [22]). In MAC we assume that the task distribution between system and user(s) can be fixed to a large extent via negotiations at the start of a KBS project where the major global functions of the system are specified. According to Moran [19] top-down system development proceeds through several layers or levels. Development starts at the *task level* with a task analysis that makes explicit which tasks user and system will be able to accomplish. Then, at the next, *semantic level*, a process model of the future system is specified that comprises the semantic objects, the semantic procedures and the control to be present in the future system. Between the task model at the task level and the process model at the semantic level there is a mapping that makes clear how the various tasks in the task level can be accomplished using the future system.

Within the context of the command-based interface, Moran introduces a few more levels in which the interface is specified. In our methodology for the analysis of cooperation these interface dependent aspects are collapsed into one level, the *communication level*. Again there is a mapping from the semantic level to the interface specification at the communication level.

In our framework we have made a vertical cut through the levels. Moran's framework is concerned with command-based systems, where the system is a passive tool and the user's tasks drive the performance. However, a KBS is considered to be a more active "agent", which necessitates the analysis of tasks of the system (i.e. the problem solver) separate from the role of these tasks and the user's tasks in the cooperation. Therefore, in the MAC framework for each layer there is a problem solving view and a cooperation view. Figure 1 provides an overview of the framework. The left hand column is concerned with the problem solving tasks when done by a single expert agent. The right hand column introduces all complications that occur when problem solving is distributed over different agents. More specifically: at the task level we distinguish between the model of the basis task (box1) and its elaboration into task distributions (box2). At the next, semantic level, we distinguish between a competence model of the domain problem solver (box3) and a *model of cooperation* (box4) that contains all additional functions -including strategic control- that serve the cooperation between domain problem solver and user. At the communication level, finally, we distinguish between the presentation or transparency of domain objects (box5) and the interaction between user and system (box6).

MAC is mainly concerned with boxes 1, 2 and 4. Box 3, the problem solver, is already covered for by the KADS methodology. The crux of the

	Problem Solving Domain	Cooperation
Task Level	**1** basis task model (task-decomposition)	**2** all task distributions that are required
Semantic Level	**3** model of problem solver	**4** model of cooperation
Communication Level	**5** transparency	**6** interaction

Figure 1: The framework for the analysis of cooperation

methodology is in box 4 where the model for cooperation is specified and empirically tested in the *user analysis*. User analysis involves empirical studies with subjects from the population of intended users and a hand-simulated system.

The specification of the communication layer depends to a large extent on the particular medium for communication that is selected, e.g. natural language, menus, windows, etc. Box 4 contains the *what* and *when* of the communication; in box 6 the *how* is specified. It should be noted that this successive distinction of what, when and how is also applied in advanced discourse planning (e.g. [2]). The communication level is not simply a further, more detailed specification of the model of cooperation. The communication level consitutes a new task distribution by itself. Communication tasks are based upon cooperation, which is particularly organized by *conventions* which may differ from medium to medium. Conventions are the pragmatics, syntaxis and lexicon of a particular medium or language. New media often imply new conventions, which have to be established and learned. For instance, there are no standards for the use of windows, icons, etc. For this reason, at the communication level the interface is constructed (designed) rather than specified in a top-down manner.

In the next sections, where we discuss the StatCons example, the communication level will be presented only incidentally. The focus is on the task and the semantic level. First the task-decomposition (box 1), then the task distributions that are required (box 2) and then the model of the problem solver (box 3). After that we will present the StatCons-1 prototype and its shortcomings. At the end we will present the model of cooperation (box 4), which extends the analysis to prevent the problems that occurred with the StatCons-1 prototype.

3 The basis task model

The aim of the analysis at the first, task level is to establish the space of task distributions for future system and user. A task distribution is a specification of which agent does which sub-task, e.g. user is assigned to 'planning the analysis' and system is assigned to 'computing statistics'. In MAC we have made a distinction between a model of the basis task and the space of task distributions defined on it. This is useful because one task model can serve as a basis for many different task distributions. The model of the basis task is a hierarchical task-decomposition in which input-output dependencies between tasks are also recorded. That is useful: when, in the task distribution, tasks are assigned to different agents, the dependencies between tasks contain all the information to derive the communication tasks that are necessary in the particular task distribution. After all, each dependency represents an object achieved in one sub-task and needed in another sub-task.

The modeling language for the basis task model is hierarchical data-flow diagraming. If, in a diagram, a task has more than one input, it means that all inputs are needed to do the task in a valid way. Figure 2 shows part of the functional task-decomposition for StatCons. The presentation style in this figure is borrowed from Connor [8].

In the StatCons case-study we have put emphasis on the task 'planning the data collection'. That is why figure 2 shows the further decomposition of this task. 'Planning the analysis' is also important, but several other researchers have been looking into statistical strategy, given a par-

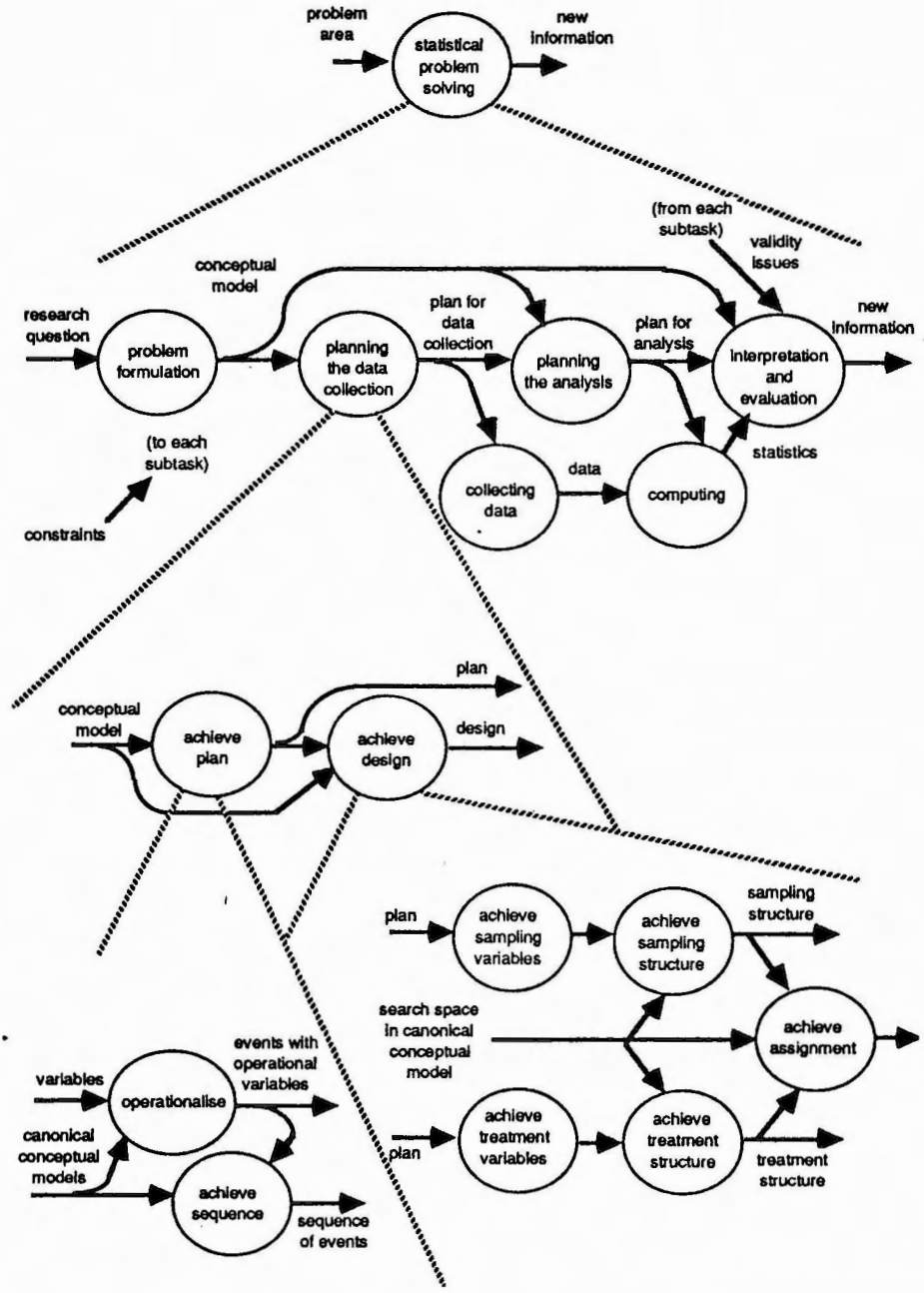


Figure 2: The task-decomposition of statistical problem solving (basis task model)

ticular statistical model, e.g. [13], [24]. For StatCons we have only paid attention to the first step in 'planning the statistical analysis', namely, the selection of a statistical model.

The task-decomposition is a functional black-box decomposition and the tasks have functional objects as inputs and outputs. These objects can be decomposed as well when task-decomposition proceeds. For example, the 'plan for data collection' is decomposed into a 'plan' and a 'design'. Below three functional objects: conceptual model, plan and design are described in more detail.

conceptual model The first sub-task in statistical problem solving is problem formulation. The output of this step is a conceptual model, a well-formulated problem. There exist paradigmatic conceptual models for well-known statistical models. These we have called 'canonical conceptual models'. A canonical conceptual model has variables as components and there can be different roles for the variables (e.g. independent and dependent variables). These roles refer to a relation between the variables (e.g. effect relation). A canonical conceptual model has a type of search space that represents the alternative models for the relation (e.g. no effect, main effects, interaction effects). This search space can be based -directly or indirectly- on the power set of the set of variables. Alternative hypotheses or models are boundary sets on this frame. To define a range of alternative models , a canonical conceptual model has a minimal and a maximal model that are both boundary sets. A canonical conceptual model is thus a qualitative abstraction from a statistical model.

plan The plan is the temporal physical system in which the data are collected. The components of a plan are events with operational variables. The structure is a temporal physical structure on the events (and the operational variables). The structure can be linear, but there may occur complicated structures with splitting or joining of temporal physical sequences.

design In so far the plan contains variables and parameters that can be controlled by the researcher, the design specifies how these will be instantiated for the set of individuals in the sample. The design comprises a sampling structure on the sampling factors, a treatment structure on the treatment factors and the assignment of units in the sampling structure to treatments in the treatment structure, under an appropriate randomization. The structures involved are relations like crossing and nesting, etc.

The description of the basis task as presented in this section (decomposition of tasks and decomposition of functional objects) provides the background to discuss task distributions (section 4) and to consider the mapping to AI problem solving methods (section 5).

4 The task distributions

The aim of the task level analysis is the specification of the negotiation space, the space of all task distributions that are required. In a fully specified task distribution each task (except communication tasks) is exclusively assigned to a single agent. The most straightforward way to distribute tasks is to assign agents exclusively to sub-tasks at some boundary set in the task-decomposition tree. Such a task distribution can be characterized as advice giving: when the system is assigned to task X the system gives advice about task X or about the output of task X. This is not always enough: below three approaches to more complicated task distributions are given. After that, the negotiation space for StatCons is outlined.

Job design It may be necessary to modify the task-decomposition, or to extend it, to arrive at sub-tasks that can be assigned to one agent exclusively. This can be characterized as job design. For example, Wittkowski has rearranged the first level decomposition of figure 2 completely to distribute sub-tasks over three agents: client, statistical expert and system [23]. First the client talks with the expert who checks the client's plan for collecting the data. After that the expert enters the constraints that are implied by the design into the system and finally the client is allowed to do the analysis using the system. For StatCons we have avoided job design.

Checking: system and user do the same task In pedagogical situations it often occurs that both pupil and expert do the same task independently and then their respective outcomes are compared, or the expert tries to follow the pupil's solution path to find if (and where) it goes wrong. We call this type of task distribution 'checking'. It involves two copies of a basis task, one assigned to the user, the other one to the system, and there is a 'compare task'. 'Coaching' and 'critiquing' are examples of this type of task distribution.

Additional ingredients To do his part of the task distribution, the user may need other ingredients besides the objects that are implied by the dependencies between sub-tasks. In pedagogical situations, these ingredients are communicated as

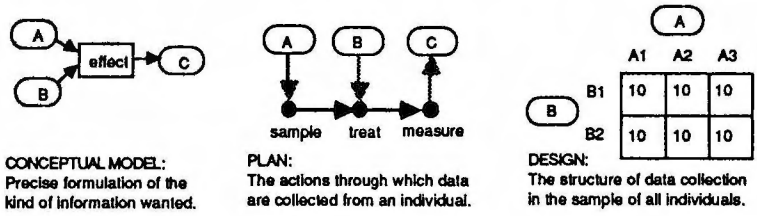


Figure 3:

well (e.g. explanations, instruction, help). In general, to do a task, other ingredients like skill and knowledge are required. If the user does not own some of these ingredients, the system can perhaps provide these. For example, the future system cannot do the task 'problem formulation', but it could assist the user by providing various knowledge items (e.g. general principles, examples). To supply these additional ingredients there will be additional communication tasks. In our methodology, the specification of these additional ingredients is postponed to the next stage in the analysis of cooperation.

Task distributions for StatCons It is clear from the protocols of real and mock-up (Wizzard-of-Oz) sessions that many different task distributions are required in the task: 'planning the data collection'. Sometimes a client himself made a complete plan but is not sure about a choice for the operationalization of a variable. Another client may want advise about sample size, etc. In all sessions we have recorded, it appears that if the client has done 'planning the data collection' all by himself, the consultant wants to check whether the client did this task in a correct way.

For StatCons and its intended users, the required space of task distributions can be characterized as follows: each sub-task of 'planning the data collection' is either assigned to the system (advice giving) or assigned to both user and system and their respective results are compared by the system (checking). The task 'select statistical model' is assigned to system (advice giving). It should be noted that in this negotiation space the task decomposition of 'planning the data collection' in figure 2 becomes dynamic and sub tasks have to be decomposed further. For example, the user has five variables that are operationalized (five times 'operationalize'). One of the five variables is operationalized as a total score of rating scales and the user wants advise about the number of points to be used for the rating scales.

5 The problem solver

We have assumed that StatCons can use the same problem solver and the same knowledge in different task distributions. In advise giving, the problem solver is used to generate a solution under constraints. In checking, the problem solver is used as a standard for monitoring the user's problem solving steps. In both cases the process can start from the conceptual model and from there it walks from choice point to choice point, considering and taking decisions at each point.

The structuring of domain knowledge in StatCons-1 is based on two 'paradigms': Subsumption hierarchies a la KL-ONE [4] which allow for classification, and associations between these hierarchies, which stand for compiled out proofs a la explanation-based generalizations. This structure allows for abstraction and refinement problem solving steps as also found in Clancey's [7] inference structure for heuristic classification.

Problems get solved by data abstraction upwards in the data hierarchy, association between a concept in the data hierarchy and a concept in the solution-hierarchy and refinement downwards in the solution hierarchy.

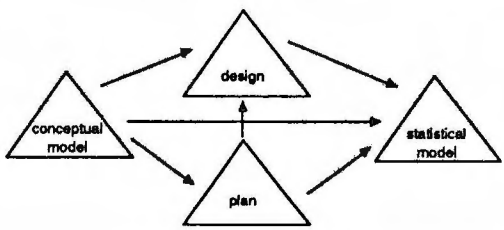


Figure 4: The structuring of domain knowledge

Figure 4 shows an overview of the structuring of domain knowledge in which KL-ONE hierarchies are combined with the model of heuristic classification. There are concept hierarchies representing paradigmatic conceptual models, plans, designs and statistical models.

The first rationale for this structuring is that the reasoning in planning data-collection is to a large extend compiled-out reasoning. The second rationale is that the objects (conceptual model, plan, design, statistical model) are decomposed far enough to be mapped onto KL-ONE concepts. The internal structure of these objects can to a large extend be represented using *component roles*. For example, the components of a plan are events and the structure is a temporal physical ordering of events. To represent a plan with a fixed structure, we can use different slots for the events that have a different place in the structure, e.g.: sampling, pre-test, treatment, post-test. For the more complicated (mathematical) structures, that are not fixed, for example the search space of a canonical conceptual model or the design structures, we use *structure roles* that are filled with boundary sets on lattice structures.

6 The StatCons-1 prototype

Starting from the analysis model of the problem solver, a prototype was designed and implemented to test the analysis model and the feasibility of a separation between analysis and design. This work was done by Schreiber and Wielemaker (Part II and part III in [12]). The result is StatCons-1, an implementation that has the same structure as the analysis model and that functions with an interface based on graphical editors.

The system fulfills the functional requirements as laid down in the analysis documentation. The actual code contains the structuring of knowledge as postulated by the analysis model of the problem solver. The design of StatCons-1 is based on the KLASS system [12], a variant of KL-ONE that appeared strong enough to provide an adequate formalization of the complex objects.

User-interface The interface is based on the direct manipulation paradigm [17]. All complex objects are visible and manipulable on the screen of a SUN work station. Through buttons the user can request the system to do a sub-task (synthesize button) that creates a complex object (plan, design, statistical model) or to check the validity of the combination of two or more complex objects (analyze button). Figure 6 shows the editors for conceptual model, plan and design (the variable editor and the window for statistical models are not shown).

Problems with StatCons-1 The implementation of StatCons-1 shows that implementation of a problem solver as described in section 5 is feasible, but the system appears unsuitable for the intended users. The prototype system comprises a problem solver that is sufficiently powerful and a direct manipulation interface that offers full functionality and maximum transparency of objects in the domain, but it appears that this is not enough. StatCons-1 is a multi-functional system. It can do different tasks or check tasks that were done by the user. A multi-functional system requires strategic control over task-selection. Within the direct manipulation paradigm, the control over the cooperation between user and system is fully in the hands of the user. The user *must* control when which tasks will be done. This is asking too much from the users in the target population. The user has a weak, insufficient conception of the task and is not able to control the system using the analyze and synthesize buttons.

7 The model of cooperation

Because of the StatCons-1 result, it became clear that the KADS methodology was one sided, with exclusive attention for the problem solver. To amend for this, we extended it with a methodology for the analysis of cooperation. Besides a model of the problem solver, the analysis stage in system development now also has to result in a model of cooperation. This model of cooperation explicates how the problem solver can be used and it can be tested in the user analysis.

The model of cooperation The negotiation space, the space of all task distributions that are required, provides the basis for the model of cooperation. In a task distribution, the communication tasks for objects (the dependencies in the basis task model) constitute the minimum flow of information needed to achieve the task. Some users may need additional ingredients (see also section 4) to do their part. Therefore, first the space of task distributions is extended with additional ingredients.

Second, the initiatives for objects and additional ingredients are specified. For each ingredient (object or additional ingredient) or, in other words, for each communication task, the knowledge engineer specifies which agent will initiate the transfer. We have called such a refined task distribution a *modality*. It is a blue-print for cooperation that states who is doing which subtask, which objects need to be communicated, which additional ingredients may be needed by the user, and who is allowed to take which initiative and the scope of these initiatives.

Third, to provide the full specification of strategic control, we have to specify how a par-

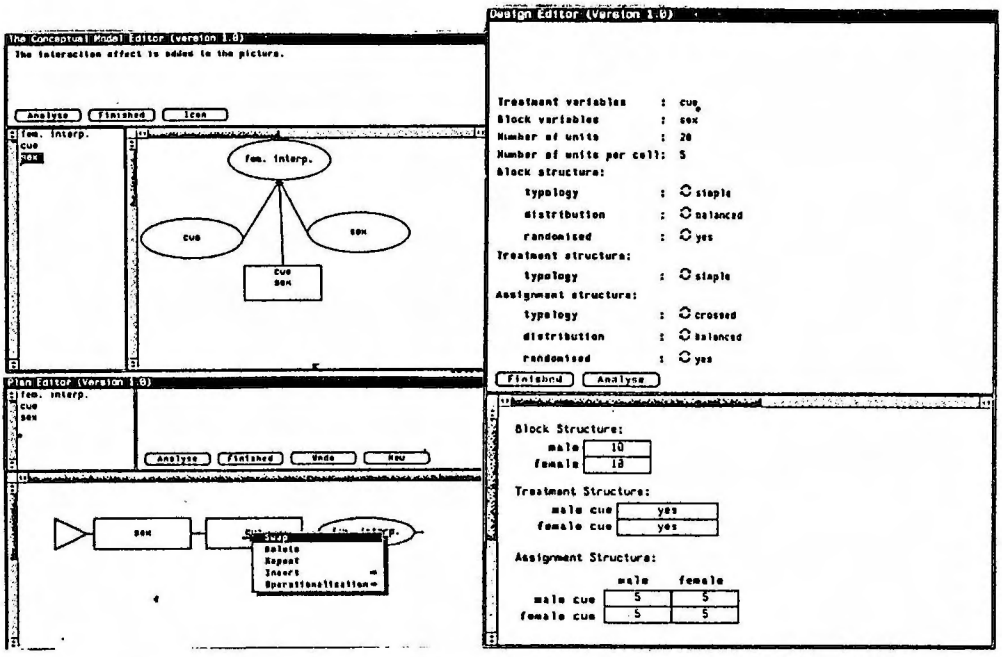


Figure 5: StatCons-1: The editors for conceptual model, plan and design

ticular modality is selected, how it is executed and how it can be modified if necessary. We call this 'cooperation management' and for this we can use a meta-level paradigm (see figure 6). This

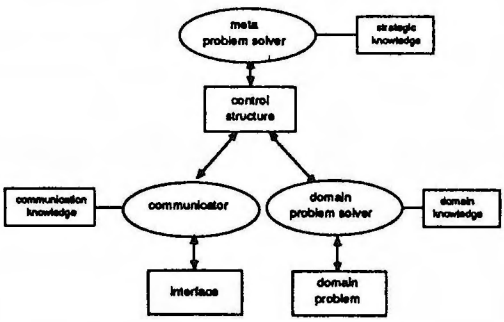


Figure 6: The meta-level paradigm

paradigm in its general form can be implemented using a segmented black-board architecture [16]. It fits well in our analysis framework: strategic control of cooperation is factored out and allocated to the meta-level problem solver. The domain problem solver thus remains a pure competence model that knows what is valid in the domain.

The meta-level problem solver, or, the cooperation manager, operates on the control structure. Let us assume the control structure contains the 'current modality' from the space of all possible modalities. The current modality is a program for cooperation that can be executed by the communicator and the domain problem solver. To ensure that the tasks as planned are accomplished, the cooperation manager must also exercise control in the cybernetic sense. Outputs of tasks must be measured to see if these conform to what was planned. If this monitoring detects a serious discrepancy (a meta problem), it may need to be diagnosed, and after that some corrective action is to be taken. (figure 7). First an attempt is made to overcome it by a local repair. This is a local insertion in the current modality and no part of the current modality is deleted. Hopefully, the local repair suffices to continue the execution of the current modality. If a local repair does not help, or if no local repair is known, the modality cannot be accomplished and needs to be (partially) retracted and replaced by a new one. This provides the abstract framework for modeling cooperation. The knowledge engineer is supposed to use this framework as an interpretation model for the particular domain and user population at hand. It should be filled in and tested in the user analysis.

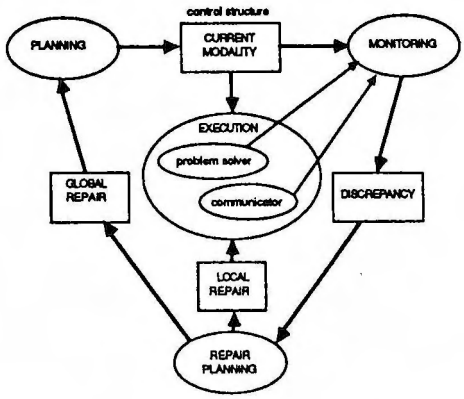


Figure 7: Decomposition of cooperation management

User-analysis The point of the user analysis is to see if the model of cooperation matches the needs of the intended users. The experimental set-up of the user analysis is shown in figure 8. It is a variant of the so-called Wizard of Oz tech-

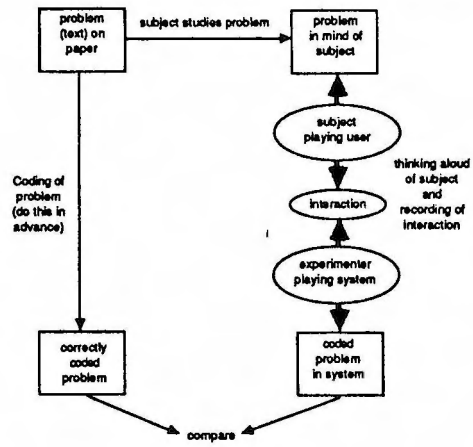


Figure 8: The experimental set-up in the user analysis

nique. The subjects are sampled from the population of intended users. A subject is presented with a problem text on paper and is asked to solve the problem using the system. The subject and the experimenter who is hand-simulating the system (cooperation manager, communicator and problem solver) are in different rooms and can exchange messages. These could be messages on paper, but it is more efficient to exchange messages over terminals (two PC's with a terminal program and a cable suffice). A logfile of the interaction can be kept and at the user side a thinking-aloud protocol is recorded. The user analysis tests whether the model of cooperation is enough and if it is not, the thinking aloud protocol provides clues about user needs that are not covered for in the model of cooperation. Hand also used terminal protocols [15], but he had a free interaction between expert and client. Here we have a model behind the interaction and the interaction is more like what we expect that a system could do.

The model of cooperation for StatCons-2 Interpreted in terms of the new framework for modeling cooperation, it appears that for StatCons-1 we have implicitly 'decided' to have no additional ingredients, to let the user have the initiative all the time and to assign all cooperation management to the user. We hypothesize that if we would have had the analysis methodology, and if we would have done the user analysis, StatCons-1 would have been designed differently. To see if this hypothesis is true we have done user analysis for a new system, StatCons-2. In total we used seven subjects (psychology students with some training in statistics) and each did two problems. In an explorative pilot stage we used three subjects and during this phase we made many ad hoc modifications to the model of cooperation. In a second confirmatory stage we used four subjects and we did not change the model of cooperation.

The space of task distributions is the same for Statcons-1 and StatCons-2, but for StatCons-2 the initiative for the communication of basis objects is assigned to system. For the pilot user analysis we made an initial list of additional ingredients:

```
introduction to the system
introduction to the task decomposition
explanation of a concept
explanation of why a piece of data is needed
explanation why a particular task needs to be
done
explanation why a task should lead to a
particular output
explanation why one output is superior over
another
change of topic
provide alternative solution
modify previously entered data.
enter data as a diagram
give advise here instead of checking me.
change to advise permanently
```

For these additional ingredients the initiative was assigned to the user, except for 'introduction to the system' and 'introduction to the task decomposition' For these the initiative was mixed, assigned to both user and system. The system would exercise these initiatives at the beginning

of the first session. A particular elaboration of the cooperation manager was used, based on a default modality. The initial planning degrades to selection of a default modality, which is checking for each sub-task in 'planning the data collection' and advise giving about 'selection of statistical model'. In the simulated cooperation manager, the modality is generated by a depth-first planner that plans the task-decomposition as shown in figure 2. As soon as this planner reaches a leaf in the plan, it is executed and monitored. If monitoring does not detect a discrepancy, i.e. if communication tasks and problem solving tasks are successful and if comparison between user's and system's intermediate solutions does not generate a discrepancy (we have checking here) the planning and executing continue until the end of the modality.

In the user analysis we have used a restricted natural language interface. To standardize and constrain the mock-up interface we prepared files with questions for data-input (mostly menu-type questions) keywords for user initiatives, canned texts and text-frames for presentation of solutions. In the pilot stage, we made many modifications to the questions and to the hand-simulation regimen. After the pilot stage, we counted the use of initiatives for additional ingredients in the interaction protocols, and it appeared that many were never used, or that they caused more problems than that they solved. Therefore, we pruned the list of additional ingredients to the following:

```
introduction to the system
introduction to the task decomposition
explanation about a specific concept.
explanation why a piece of data is needed
give advise here instead of checking me
change of topic
modify previously entered data
```

We also made a more detailed specification for the system behavior after monitoring a discrepancy, and we added text-frames for the two explanations on the new list of additional ingredients. This refined model of cooperation then was hand simulated in eight sessions with four new subjects. The interaction protocol of one session was lost, leaving seven sessions to evaluate. Five of the seven sessions resulted in a correct description of the problem on paper. The two incorrect descriptions in the remaining two sessions had incorrect conceptual models. From this result and from clues in the thinking-aloud protocols there appear to be two user needs that were not covered for. The following two additional ingredients can probably provide the remedy:

```
support in formulation of the conceptual model
overview of all data on the problem
```

The most important outcome of this user analysis is that users do not need most of the additional ingredients we had listed a priori, and that there were two user needs we did not think of in advance.

The results of the user analysis suggest that effective cooperation is feasible if the system takes initiative for the communication of basis objects and if the user is provided with initiatives for the additional ingredients. Yet we cannot draw definite conclusions since we only had four subjects and two problems.

We want to repeat the user analysis with a more extensive set of problems and a larger sample of intended users. We would like to be certain that the model of cooperation need not be concerned with detailed user modeling. User modeling may be needed to schedule system initiatives for additional ingredients. This would involve complications we rather not have when we do a redesign and implementation of StatCons, and if this is needed, we would like to know beforehand.

8 Summary and concluding remarks

MAC is part of the KADS methodology and is aimed at the specification of the cooperation between user and KBS. The models of cooperation are the constraints for designing the user interface.

MAC starts with a task analysis of the operational context in which the prospective KBS is to be used. A further refinement via distributing tasks over user(s) and KBS(s) leads to a detailed specification of the various cooperation 'regimes' that are required.

MAC aims at collecting all the information that is necessary to design a cooperative system. MAC includes a verification of this information by user analysis. We have illustrated the use of MAC in the StatCons domain and we did a small-scale user analysis with the model of cooperation for StatCons.

User analysis is not the collection of arbitrary teletype protocols. User analysis presupposes a model of cooperation. The model is used to guide data collection and it helps the analysis of the verbal data thus obtained. Without a model, without knowing what to look for in the data, the protocols are hard to analyze.

In this way a minimal set of additional ingredients and initiatives can be obtained that are required for the worst-case user. More importantly, the user analysis helps to assess the *feasibility* of an *operational* expert system for a *specific user population.*

The focus in this paper has been on the worked-out example of knowledge acquisition

aimed at fitting the prospective KBS to the needs of the operational environment, resp. of the user. We did not go into details of the Modality framework nor at the theoretical background. These issues are discussed in [10].

Acknowledgements Many people have contributed to the development of MAC and StatCons. Ton de Jong, Paul Lim, Guus Schreiber, Dick van de Vlugt, Jan Wielemaker and Bob Wielinga

The research reported is part of the KADS project which is partially funded by the Commission for the European Communities ESPRIT Programme, as project number 1098. The partners in the KADS project are STC plc (GB), SCICON Ltd (GB), NTE GmbH (D), Cap Sogeti Innovation (F), University of Amsterdam (NL), and KBSC of Touche Ross (GB).

References

[1] E. E. Adam and R. J. Ebert. *Production and operations management.* Prentice Hall, London, 1982.

[2] J. Allen. *Natural Language Understanding.* Benjamin Cummings, Menlo Park, CA, 1987.

[3] A. H. Bond and L. Gasser, editors. *Readings in Distributed Artificial Intelligence.* Morgan Kaufman, San Mateo, CA, 1988.

[4] R.J. Brachman and J.G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9:171–216, 1985.

[5] J. Breuker, B. Wielinga, M. van Someren, R. de Hoog, G. Schreiber, P. de Greef, B. Bredeweg, J. Wielemaker, J-P Billault, M. Davoodi, and S. Hayward. *Model Driven Knowledge Acquisition: Interpretation Models.* Esprit Project P1098, Deliverable D1 (task A1), University of Amsterdam and STL Ltd,, 1987.

[6] J. A. Breuker and B. J. Wielinga. Model driven knowledge acquisition. In P. Guida and G. Tasso, editors, *Topics in the design of expert systems*, North Holland, Amsterdam, 1989.

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

[8] D. Connor, editor. *Information System Specification and Design Road Map.* Prentice-Hall Inc., Englewood, New Jersey, 1985.

[9] P de Greef. *Cooperative Statistical Problem Solving.* Paper presented at the Second International Workshop on AI and Statistics, SWI, University of Amsterdam, 1988.

[10] P. de Greef, J. Breuker, and T. de Jong. *Modality: an analysis of functions, user control and communication in knowledge-based systems.* Esprit Project P1098, Deliverable D6 (task A4), University of Amsterdam, 1988.

[11] P. de Greef, J. Breuker, G. Schreiber, and J. Wielemaker. StatCons: knowledge acquisition in a complex domain. In *Proceedings ECAI'88*, Munich, 1988.

[12] P. de Greef, G. Schreiber, and J. Wielemaker. *The StatCons Case Study.* Esprit Project P1098, Deliverable E2.3 (experiment F2), University of Amsterdam, 1987.

[13] W.A. Gale, editor. *Articial Intelligence and Statistics.* Addison-Wesley, 1986.

[14] M. P. Georgeff. A theory of action for multi agent planning. In , editor, *Proceedings of the AAAI 1986*, pages 121–125, Morgan Kaufman, Los Altos CA, 1984.

[15] D. J. Hand. Patterns in statistical strategy. In W. A. Gale, editor, *Artificial Intelligence and Statistics*, pages 355–387, Addison Wesley, Reading, MA, 1986.

[16] B. Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26(3):251–321, 1985.

[17] E.L. Hutchins, J.D. Hollan, and D.A. Norman. Direct manipulation interfaces. In D.A. Norman, editor, *User Centered System Design*, Lawrence Erlbaum, Hillsdale, N.J., 1986.

[18] M. J. S. Rosenschein, M. Ginsburg and M .R. Genesereth. Cooperation without communication. In , editor, *Proceedings of the AAAI 1986*, Morgan Kaufman, Los Altos CA, 1986.

[19] T. P. Moran. The command language grammar: a representation for the user-interface of interactive computer systems. *International Journal of Man-Machine studies*, 15:3–50, 1981.

[20] D.A. Norman and S. W. Draper. *User Centered System Design.* Lawrence Erlbaum, illsdale, NJ, 1986.

[21] M. E. Pollack. *Inferring Domain Plans in Question Answering.* PhD thesis, University of Pennsylvania, 1986.

[22] C. J. Stuart. An implementation of a multi agent plan synchronizer. In M. G. Georgeff and A. M. Lansky, editors, *Reasoning about actions and plans: proceedings of the 1986 workshop*, pages 161–188, Morgan Kaufman, Los Altos CA, 1987.

[23] K.M. Wittkowski. Generating and testing statistical hypotheses: strategies for knowledge engineering. In R. Haux, editor, *Expert Systems in Statistics*, pages 139–154, Fisher, Stuttgart, 1986.

[24] D. E. Wolstenholme and J.A. Nelder. A front end for GLIM. In R. Haux, editor, *Expert Systems in Statistics*, pages 155–177, Fisher, Stuttgart, 1986.

ASKE: TOWARDS AUTOMATIC KNOWLEDGE ACQUISITION

Jitu Patel
Human Cognition Research Laboratory
The Open University
Milton Keynes, MK7 6AA, UK
e-mail: jm_patel@uk.ac.ou.vaxa

ABSTRACT

This paper describes work in progress towards the automation of the knowledge engineering process. The ensuing system for knowledge acquisition, called ASKE, incorporates expertise modeled on our experiences of building expert systems. The system carries out knowledge acquisition in two stages. In the first stage, ASKE interviews the expert to build a task model of the domain. It does this by using previous cases, in a similar domain, as exemplars of what knowledge should be contained in the task model. In the second stage, the task model is used to elicit the problem-solving expertise from the domain expert. The methodology draws its power from the representational scheme of templates that ASKE uses to encode domain knowledge. The merit of this approach to knowledge acquisition is that the system can be used to build prototype expert systems for solving any analysis task.

1 INTRODUCTION

Knowledge Acquisition (KA), the transfer of problem-solving knowledge from a domain expert to a computer program, has been described as a major obstacle in the development of expert systems (Feigenbaum, 1977), and is a very time consuming activity (e.g., McDermott, 1982; Smith and Baker, 1983). One of the reasons for a lengthy development cycle for expert systems is that the knowledge engineer has to spend time to learn about the domain so that s/he can interact with the expert to effectively elicit, interpret and organise the problem-solving expertise. A solution to the problem is to let the expert build the system himself. This, in fact, is the guiding principle for work in KA:

> "The ultimate design goal for knowledge acquisition is to allow the expert to encode his own knowledge directly into the computer, removing the role of the knowledge engineer from the knowledge acquisition phase." (Weiss & Kulikowski, 1984; p.62)

In the last few years a growing number of researchers have worked towards this aim of producing KA systems that can be used by domain experts to produce expert systems. Work in automatic KA has progressed in two distinct but complementary directions: (1) systems which use inductive techniques for learning from examples (e.g., Rouveirol and Manago, 1988); and (2) systems which use interviewing methodology based on either problem-solving methods or psychological techniques (e.g., Boose, 1988). The research presented in this paper takes the second direction.

A very large number of systems have been developed within the interviewing paradigm. On the basis of their approaches, most of these systems can be classified as technique-based, knowledge-based or some mixture of both. We will consider each of the approaches on the basis of the following success

criteria: (1) the scope of applicability or breadth of the system; and (2) the strength of the system's problem-solving expertise or depth of the system.

In the technique approach, researchers have investigated KA by implementing psychological techniques for knowledge elicitation. The best examples of this line of research are Boose (1984) and Shaw and Gaines (1987). Their respective systems, ETS and KITTEN, use the repertory grid methodology to acquire knowledge from the expert. A strength of these systems is that they can be used to solve any task for which solutions can be pre-enumerated. Their weakness is that they extract relatively weak problem-solving expertise. While this approach has been marked by its domain independent strategies, the second line of research, the knowledge approach, has looked towards using domain-specific knowledge for KA. Systems have been developed (e.g., Kahn et al., 1985; Eshelman and McDermott, 1985) which use problem-solving methods to drive the KA process. This class of systems have succeeded in eliciting stronger problem-solving expertise at the expense of limited scope. Thus, the two approaches perform differently on the breadth-depth scale, and we will refer to this as the breadth-depth problem.

The mixed approach, as its name suggests, includes both the technique and knowledge components. The approach normally involves the development of a conceptual model which is then used for driving knowledge elicitation. Essentially, within this perspective, KA is seen as a two-stage process. In the first stage, the expert is guided towards the development of a task model for the domain. In the second stage, the model is used to elicit the problem-solving knowledge from the expert. This methodology has been successfully implemented in the KNACK (Klinker, et al., 1987) and PROTEGE (Musen, 1988) systems. Evaluation of these systems suggests that the modeling approach can be utilized for designing a system which has both breadth and depth. Our aim in this paper is to present a methodology, based on the modeling approach, which explicitly addresses the problem of breadth-depth tradeoff. The methodology is implemented in our system, ASKE (Automatic System for Knowledge Engineering).

The rest of the paper is organized as follows. In section 2, we present the ASKE system. The discussion is related to the ASKE methodology and the representational details of the system. In section 3, a scenario of an ASKE session is presented to illustrate the methodology. In section 4, we discuss the merit and scope of the methodology.

2 THE ASKE SYSTEM

ASKE is an automatic system for KA. It can be used by experts to build prototype expert systems for solving analysis problems. The ASKE methodology is based on the assumption that the most difficult part for the expert is the formalization of his/her domain knowledge. The system is, therefore, built on the premise that KA process can be quickened by helping the expert organize his/her domain knowledge. This intuition is based on the experience we gained at building an expert system for interpreting archaeological sites (Patel and Stutt, 1988). The hardest aspect of the system development was the recognition of the important concept categories for the domain and how these were inter-related. Once we had the model of the domain, the knowledge elicitation became just a routine task.

How should the domain knowledge be organized to make it more conducive to the KA activity? In the ASKE methodology we were guided by the interaction problem (Bylander and Chandrasekaran, 1987). According to the interaction problem, how knowledge is represented is dependent on how it will be used. In the ASKE methodology, the inference strategy of heuristic classification (Clancey, 1985) is used as the paradigm for organizing domain knowledge. Thus, problem-solving knowledge is seen as consisting of two main parts: the data (or the abstract problem statement) and the solution. The problem-solving expertise is expressed as the ability to make heuristic associations between the data and the solution categories.

2.1 BASIC ASSUMPTIONS

In the design of ASKE, two main assumptions were made. The first assumption is about the type of problem the ensuing expert system will be solving. ASKE is geared for developing systems that do analysis, that is, the problem solver selects a solution from a set of pre-enumerated solutions. This implies that the problem solver has a knowledge of the data and solution categories. It is further assumed that the expertise lies in being able to identify the heuristic association between the categories

The second assumption is that experts can effectively learn to organise central concepts in their domains from examples of organized knowledge bases in other domains. Research in analogical problem solving has shown the people routinely borrow knowledge from other domains to learn more about their own domain (e.g., Keane, 1988).

2.2 METHODOLOGY

The ASKE methodology for KA is based on the modeling approach. The KA takes place in two stages, as shown in Figure 1. In the first stage, a task model of the domain in developed. The model building is guided by two bodies of knowledge that ASKE has: the knowledge of possible problem types and an exemplar for the domain. The former provides information about the kind of knowledge that is to be obtained from the expert. For example, what are the main data and solution categories for the problem type. Often, experts find it difficult to think in this top-down fashion. To assist them, an example of a similar, already built model is presented.

In the second stage, the task model is used to guide the acquisition of the concept hierarchies and heuristic associations between the hierarchies. Once again, the exemplar has a role to play. The knowledge of how concept hierarchies were constructed in a similar domain can be very useful. For example, aids infections can be classified in terms of the site of infection or categorized by the type of agents. The expert can be aided in decision making from the knowledge of how infections were categorized in a similar system which diagnosed infectious diseases. The use of an exemplar to assist the structuring of concept hierarchies is, however, not very well developed in ASKE.

At the end of stage two, ASKE generates an if-then rule for every heuristic association. The expert is provided with facilities for editing the rules. The final output is an intermediate level rule which can be mapped onto the KEE[1] or the KEATS[2] rule system.

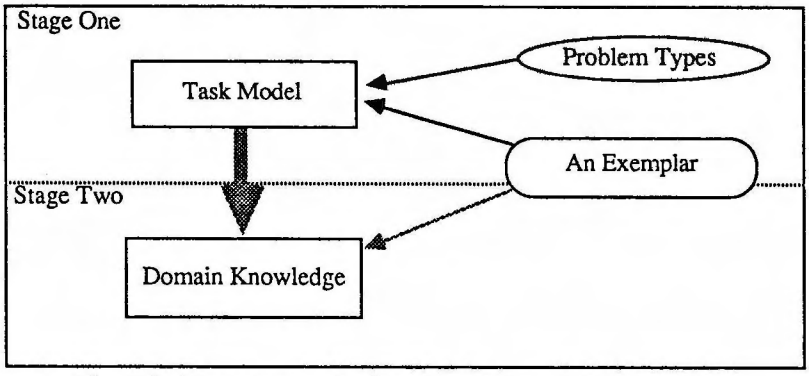


Figure 1 A two stage model for automatic knowledge acquisition

[1] Knowledge Engineering Environment (v 3.1) - © Intellicorp (1987)

[2] Knowledge Engineer's Assistant (Motta, et al., 1989).

2.3 REPRESENTATION

There are two central features to the ASKE methodology: the task model and usage of past cases as exemplars. These features are represented in templates. A template is a data structure built on top of the frame language. We begin this section by describing the notion of templates, as used in ASKE, and then look at the two features in greater detail.

2.3.1 Templates

ASKE uses the notion of a template to store knowledge. There are four main types of templates, which are classified on the basis of the function they serve. A summary of the template types is given in Figure 2.

Template	Contents	How Obtained	Use
GTEMP	Knowledge about domain hierarchy	Interaction with the expert	Selecting an RTEMP
ATEMP	General domain knowledge	Interaction with the expert	Building new knowledge bases
RTEMP	Domain-specific knowledge	Extracted from knowledge base	An exemplar for the new knowledge base
WTEMP	New knowledge base	Interaction with the expert	Holding new knowledge

Figure 2 Template Types

The most basic template is referred to as *general template* (GTEMP). A GTEMP is used for storing general information about the domain. This includes information about the expert and the domain classification. The latter is used for selecting a previous knowledge base to act as an exemplar for guiding the development of the task model. An *acquisition template* (ATEMP) contains general domain knowledge that includes main concept types and their inter-relationships. The ATEMP corresponds to a task model for the domain. It is used to guide ASKE's interaction with the expert. For the specialist areas for which knowledge bases have been developed, ASKE will have a *reference template* (RTEMP). An RTEMP consists of domain-specific knowledge which is used as an example of how a new knowledge base may be developed. A *working template* (WTEMP) is created from the ATEMP at the start of the knowledge elicitation session. This template is subsequently filled up by the new knowledge.

The templates are organized hierarchically. Roughly, the templates can be mapped on to the domain classification shown in Figure 3. The GTEMP contains information at the domain level. The ATEMP is equivalent to the task level and the WTEMP to the specialist area. The RTEMP is derived from the WTEMP. The Figure is however not to be taken literally. For example, it reads that the there is only one ATEMP for the specializations of Foot Problems and Aids Infections. This is strictly not true because though at a general level the data and solution categories are the same, the names of the categories are different. For both, Foot Problems and Aids Infections the data is the same (i.e., symptoms). However, the two specializations use different jargon to refer to the cause for the presence of symptoms. Whereas the solution category for Foot Problems is *problems*, for Aids Infection it is *infections*. It must be noted that the difference in names, albeit jargon, is a characteristic of expertise and cannot be separated from it.

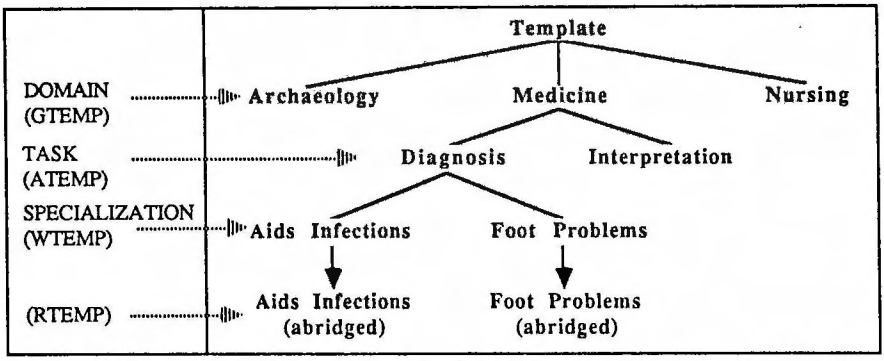


Figure 3 Organization of Templates

2.3.2 The Acquisition Template

The basic template for acquiring knowledge is the ATEMP, the initial task model. The model includes information on the main concept categories for the task and how these concepts are related to each other. ASKE has knowledge about different problem types for classification problem solving. Adopting a system approach (Clancey, 1985) rather than the object approach (Hayes-Roth et al., 1983), we have categorized problem types in terms of their input/output behaviour. A summary of the task categorization is given in Figure 4. The above four problem types are taken from Boose's (1988) categorization of analysis task. The tasks share the same inference strategy, however they differ in the kind of knowledge that the system takes as a description of the problem (data or input) and what constitute a solution (output). For example, diagnosis in the medical domain will have *symptoms* as the data category and *disease* as the solution category[1].

Task Type	Data Type	Solution Type	Comment
Interpretation	Data	Description	There can be more than one of each data and solution types
Diagnosis	Signs of Malfunction Object	Causes of Malfunction	The object shows signs of malfunction.
Repair	Fault Object	Repair of fault	The object has fault
Classification	Set of Properties	Set of Objects	A straight mapping between data and solution types

Figure 4 Problem Types for Analysis

2.3.3 The Reference Template

RTEMPs play a important role in the acquisition of the ATEMPs and also in the main knowledge elicitation session. RTEMPs are extracted from developed knowledge bases. The main types of knowledge that an RTEMP would contain includes knowledge about the main concept categories for the domain, a description of the task, and one-level deep hierarchies for each of the main concepts.

[1] This is true for most of medical diagnosis, but not all. For instance, in the diagnosis of Aids Infections the solution category is called *infection* rather than *disease*.

The RTEMPs are subsequently used as exemplars on the basis of which new applications may be developed. This usage of RTEMPs is like the interviewing strategy in ROGET (Bennet, 1983), which used knowledge of existing expert systems to acquire the conceptual structure. The novelty of RTEMPs is that they are selected at run-time.

2.4 ARCHITECTURE AND INTERFACE

ASKE is implemented within the KEE environment. The interaction takes place in two windows, corresponding to the two functional units of ASKE: the Notebook and the Rulemaker. The first window, shown in Appendix A, is where most of the earlier session occurs. All of the stage one of KA is done via question-answering. The first half of the second stage is done via graphical input. The graphical input is provided by the Notebook facility.

The second functional unit is the Rulemaker. The interface is shown in Appendix C. Once the heuristic associations between concept categories are identified, ASKE automatically generates intermediate level rules, which are displayed in the Rulemaker. The Rulemaker provides facility for editing rules and also creating new rules directly.

3 KNOWLEDGE ACQUISITION WITH ASKE

ASKE does KA in four main stages: 1) Selection of the RTEMP; 2) Building an ATEMP for the domain; 3) Eliciting domain knowledge; 4) Constructing rules. In this section, we will illustrate how ASKE processes information by going through the main steps in the development of a prototype system in the nursing domain. Because ASKE uses a mixed interface strategy of question-answering and graphical input, the depicted dialogue is a reconstructed one. Screen snapshots for part of the session are given in Appendices A-E.

A daily chore for the nursing profession, and one which is particularly conducive to the expert system technology, is to diagnose patient problems (Jones, 1988). For example, we will consider the problem of pressure sores. Typically, the task involves identifying the at-risk pressure sore patients and drawing up care plans (i.e., course of treatment) for each of them.

3.1 DOMAIN CLASSIFICATION

The aim of the first stage of KA is to acquire the domain classification and select an RTEMP, which will subsequently used to guide the building of the task model. The knowledge is held in the GTEMP. The interaction is mainly question answering.

....
User is asked to select a domain from the list of domains that ASKE knows about.[1]
What is your domain? [Enter a number]
1 Archaeology 2 Medicine 3 Materials Science 4 Motor Mechanics 5 Other
=> 5
Enter name of domain
=> Nursing
The user is given an explanatory note of what each of the problem types.
What is your task? [Enter a number]
1 Interpretation 2 Selection 3 Monitoring 4 Diagnosis 5 Repair 6 Other
=> 4
What is your specialist area?

[1] Italics are metacomments which don't appear in the interaction.

=> pressure sores
....

From the above information, ASKE proceeds to select a reference template. The basic inference is as follows:

Case 1: domain, task-type and area match -> pick area template
Case 2: domain, task-type match -> pick different area template
Case 3: no match -> find domains with task type. ask user to pick area from set of RTEMPs

The third above is satisfied, and the session continues:

Please select an RTEMP from the following. The RTEMP will be used to guide the development of the task model. Your criteria for selection should be based on your familiarity with the domain. In each of the areas, the problem type of diagnosis was used.
1 aids infection (medicine)
2 foot problems (medicine)
3 engine problems (motor mechanics)
=> 1
The RTEMP derived from the Aids Infection knowledge base is selected.

3.2 BUILDING AN ATEMP

The building of the task model is the most difficult stage of the entire KA process. At this stage, the expert is expected to lay down the main concept categories for the domain. Once the categories are identified, half the work towards the development of the knowledge base is done. The problem stems essentially from the fact that the expert is forced to reorganise his/her domain knowledge. To assist the expert, we use previous cases as exemplars.

The aim is to identify the main concept types for the diagnosis of pressure sores. A typical diagnostic task as the following knowledge:

Data:	signs of malfunction
	object (which is defective)
Solution:	causes of malfunction

For example, in the diagnosis of aids infections, the expert identified the following concepts:

Data:	symptoms (typical symptoms for the aids patient)
	patient (person suffering from aids)
Solution:	infections (types of infections to which aids patients are prone)

The aim of the project was to identify the infection on the basis of given symptoms.

What is the data type for pressure sores
=> signs of pressure sores
Could please give a short description of signs of pressure sores.
=> typical symptoms for the pressure sore patients
Do you think information about the object of diagnosis is important? [Y/N]
=> y
What is the object name?
=> patient
....
What is the solution type for pressure sores
=> care plans
....

3.3 ELICITING DOMAIN KNOWLEDGE

Once the concept types area identified, the user is asked to fill in the categories via the graphical interface, as shown in the Appendices A and B. In the left window, the user is given instructions. The user can input attributes as well as possible values for the attributes. Next stage is for the user to assign heuristics between the data and the solution types. At this stage, the knowledge base will contain the following domain knowledge.

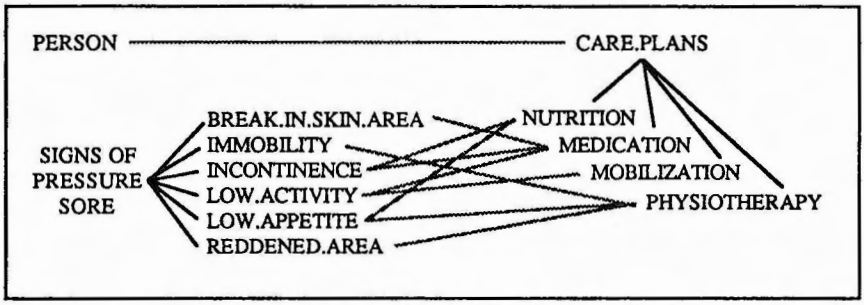


Figure 5 Concept categories and their inter-relationships

Figure 5 shows the concept categories and their inter-relationships. The shadowy lines are the heuristic associations. Note that if a relationship is identified at the top level, all the descendant nodes will inherit the relationship as well. When the user quits the session, any unaccounted for concept will be queried about. For example, if the *immobility* had not been associated with any concept in the *care.plans*, the user will be prompted:

> Jitu, my analysis show that the concept of immobility is not associated with any other concept. Could you please identify a concept from the care.plans category which is associated with immobility.
> => physiotherapy

3.4 CREATING RULES

From the heuristic association between concept categories, ASKE generates if-then rules. This rules are presented in the Rulemaker window, as shown in the Appendices C and D. The user is asked to edit the rules. The editing includes merging, adding and deleting rules as well as amending the condition and action part of the rules. Finally, the rules are translated into an intermediate-level form.

For example, the care plan of the physiotherapy is heuristically associated to the signs of immobility, inactivity and incontinence. The patient data is inherited from the care plan. The associations are translated into the if-then rule format. Physiotherapy is the action part for the following conditions:

1) patient (previous illness, age)
2) immobility (cause)
3) low appetite
4) reddened area

The user can merge the patient and immobility information to give the following:

IF patient
previous illness (serious none)

age (<50 50-60 60-70 70-80 >80)
immobility
cause (equipment splintage coma)
THEN physiotherapy

After editing the condition part of the rule, the rule is translated into an intermediate-level:

IF previous illness of patient is none
age of patient is <50
signs.of.pressure.sore is immobility
cause of immobility is splintage
THEN careplans is physiotherapy

This intermediate level rule can be mapped onto a KEE or KEATS rule directly.

4 DISCUSSION

The ASKE methodology has been successful in cutting through the breadth-depth problem. The domain-independent strategy for the acquisition of the task model gives ASKE a wide scope of applicability. At this stage, the system acts very much like those built on psychological techniques (e.g., ETS). Once the task model is obtained, the system functions in the manner of systems which use domain-specific knowledge for KA (e.g., MOLE). Thus, compared to the ETS class of systems, ASKE system has more depth (i.e., greater functionality). Furthermore, the scope of ASKE is much wider compared to the MOLE class of systems.

On the whole, the ASKE system is suitable for developing prototype systems solving analysis problems. The system has, therefore, a very wide applicability. Because ASKE, in its present stage of development, only produces intermediate-level rules, it is difficult to evaluate the power of the system with respect to the problem-solving expertise. Thus, on the issue of depth, ASKE can be said not to perform as well as MOLE. However, with future extensions to the system, in the form of a knowledge-base consistency checker, it is expected to reverse the state of affairs.

4.1 REPRESENTATION

The power of the ASKE system stems from how knowledge is represented in the system. The template approach to representation an easy way of encoding and retrieving knowledge. The approach is however not without faults. For instance, we have not touched the issue of efficiency, which is of prime importance in any real-time application. The ASKE methodology requires the storage of all system built using the ASKE system, as the previous cases are used as exemplars for the development of new systems. Over prolonged usage, it can be seen that the problem of space, and time spent in selecting an RTEMP from a wide selection will become a problem issue.

The strongest feature of the ASKE methodology, the ATEMP, is unfortunately the least developed one. The ATEMP can only be used for encoding knowledge about the four problem types discussed earlier. The task modeling would break down if a new problem type is introduced. This is really a fault of present implementation, and not of the methodology.

A further problem with the ATEMP concerns the specification of the problem types. The representation of the problem types is only geared for the acquisition of shallow domain knowledge. It is not clear how the template approach would succeed in acquiring knowledge about reasoning from first principles.

4.2 INTERFACE

One of the main issues in automatic KA is that of interface. If the machine is to elicit knowledge, then it must have a good interface otherwise there would be very little advantage in using the system if the expert has to spend a good deal of time learning it. In the design of ASKE, a great deal of effort was spent in designing the interface. For example, a graphical interface was adopted for the encoding of concept hierarchies and identification of heuristic associations between concept categories. The graphical input has the advantage over the question-answering in that it is user driven. The drawback is that the user has to decide how best to organize concept hierarchies.

4.3 BUILDING LARGER APPLICATIONS

ASKE has, as yet, not been used for building large applications. But, from its performance at building small prototypes, ASKE is predicted to work as well in bigger applications provided the proposed application is for solving analysis tasks. In its present state, ASKE is best not used for developing large knowledge bases for the system does not carry out consistency checking of the rule base. What is amiss in ASKE is a performance component, as is present in the MOLE system.

5 CONCLUSION

The ultimate áim of any KA system is to be able to produce systems solving a range of problems at the expert level. The ASKE system has been designed with this explicit aim. The two-stage model of KA shows that the breadth-depth problem can be resolved. The methodology has been used to develop a number of prototype systems, including diagnosing aids infection (Medicine), interpreting settlement sites (Archaeology), and selection of materials (Material Science).

ACKNOWLEDGEMENTS

I would like to thank the members of the KEG group for their comments; Jenny Jones for her domain expertise; Mark Keané and Tim Rajan for reviewing an earlier draft of this paper; and Sylvie Bernard for her understanding. This research is supported by an Open University Studentship.

REFERENCES

Bennet, J. S. (1983) ROGET: a knowledge based consultant for acquiring the conceptual structure of an expert system. Stanford Heuristic Programming Project, Memo HP-83-24.

Boose, J. H. (1985) A knowledge acquisition program for expert systems based on personal construct psychology. *International Journal of Man-Machine Studies*, *23*, 495-525.

Boose, J. H. (1988) A survey of knowledge acquisition techniques and tools. *Proceedings of the Third Knowledge Acquisition for Knowledge Based Systems Workshop*, Banff, Canada.

Bylander, T. and Chandrasekaran, B. (1987) Generic tasks for knowledge-based reasoning: the "right" level of abstraction of knowledge acquisition. *International Journal of Man-Machine Studies*,, *26*, 231-243.

Clancey, W. (1985) Heuristic Classification. *Artificial Intelligence*, *27*, 289-350.

Eshelman, L. and McDermott, J. (1986) MOLE: A knowledge acquisition tool that uses its head. *Proceedings of the Fifth National Conference on Artificial Intelligence*, 950-955.

Feigenbaum, E. (1977) The art of artificial intelligence: I. Themes and case studies of knowledge engineering. *Proceedings of the Fifth International Joint Conference of Artificial Intelligence*, 1014-1029.

Hayes-Roth, F., Waterman, D. and Lenat, D. (eds). (1983) *Building Expert Systems*. Addison-Wesley, Reading, Massachusetts.

Jones, J. (1988) Clinical reasoning in nursing. *Journal of Advanced Nursing, 13*, 185-192.

Kahn, G., Nowlan, S., and McDermott, J. (1985) Strategies for knowledge acquisition. *IEEE Transactions on pattern analysis and machine intelligence, 7*, 511-522.

Keane, M. (1988) *Analogical Problem Solving*. Ellis Horwood Ltd., Chichester, UK.

Klinker, G., Bentolila, J., Genetet, S., Grimes, M., and McDermott, J. (1987) KNACK - report-driven knowledge acquisition. *International Journal of Man-Machine Studies, 26*, 65-79.

Marcus, S. (1988) *Automatic knowledge acquisition for expert systems*. Kluwer Academic Publisher, Boston.

McDermott, J. (1982) R1: A rule-based configurer of computer systems. *Artificial Intelligence, 19*, 39-88.

Motta, E., Rajan, T., and Eisenstadt, M. (1989) Knowledge acquisition as a process of model refinement. *International Journal of Man-Machine Studies*. (Also available as: HCRL Technical Report no. 40, The Open University, Milton Keynes, UK.)

Musen, M. (1988) An editor for the conceptual models of interactive knowledge acquisition tools. *Proceedings of the Third Knowledge Acquisition for Knowledge Based Systems Workshop*, Banff, Canada.

Patel, J. and Stutt, A. (1988) KIVA: an archaeological interpreter. HCRL Technical Report no. 35, The Open University, Milton Keynes, UK.

Rouveirol, C. and Manago, M. (1988) Widening the knowledge acquisition bottleneck. *Proceedings of the Third Knowledge Acquisition for Knowledge Based Systems Workshop*, Banff, Canada.

Smith, R. and Baker, J. (1983) The Dipmeter Advisor System. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence.*, 122-29.

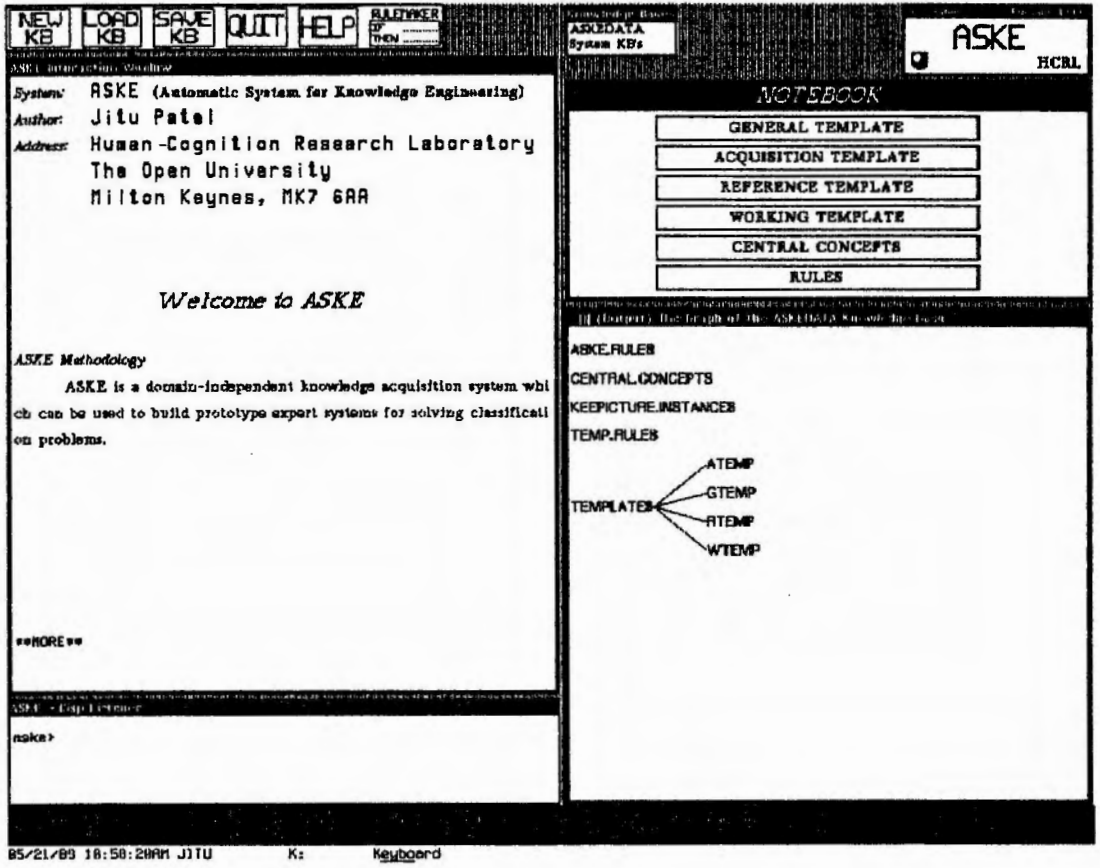
Weiss, s. and Kulikowski, C. (1984) *A practical guide to designing expert systems*. Rowman and Allanheld, New Jersey.

APPENDIX A

The ASKE Interface

The main session takes place in the Interaction Window (left). The session starts by clicking on the New KB icon (top left). The first two stages of KA are via question-answering. The final two take place through graphical interface. During these later stages, the Interaction Window is used as for displaying instructions about inputing data.

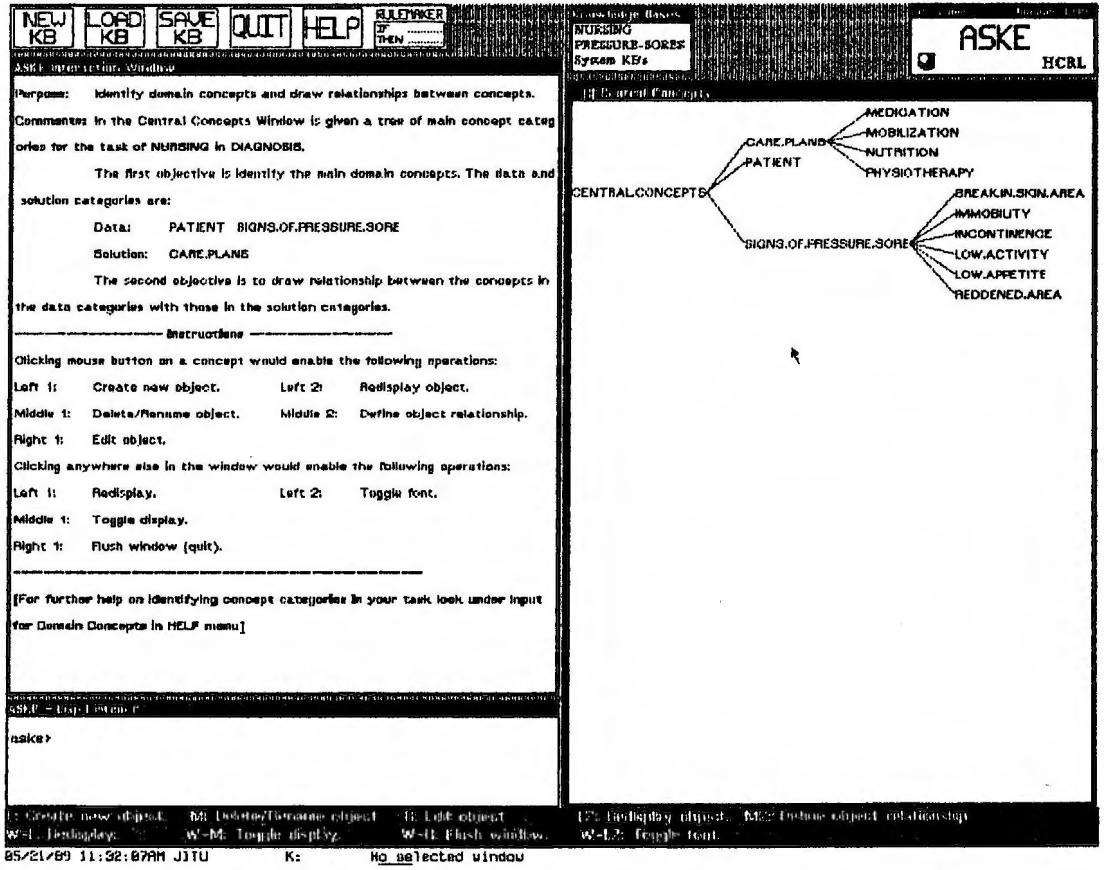
The Notebook (top right) has 6 pages which can be opened by clicking on them. The Notebook pages, when clicked, open up a graphical window and a set of instructions for input/output actions are displayed in the Interaction Window.



APPENDIX B

Eliciting Concept Hierarchies

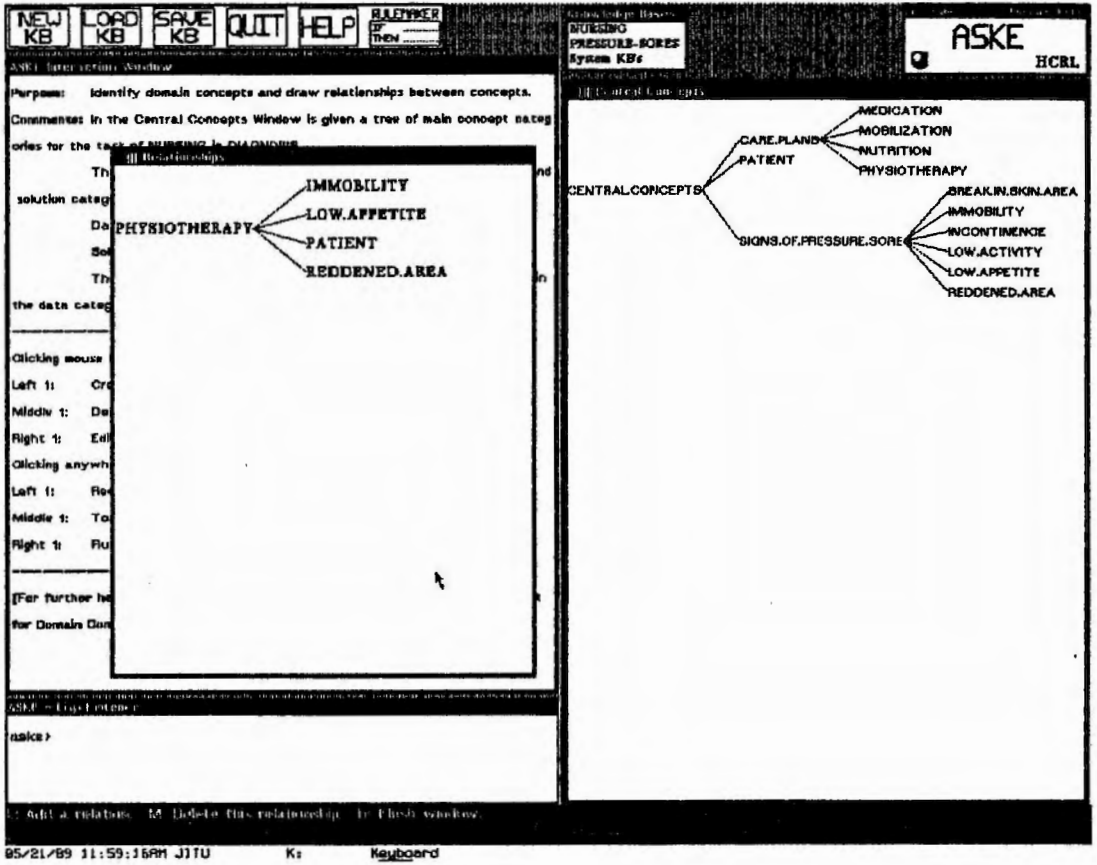
The Central Concepts Window (right) is opened at the start of the third stage of KA, or when the user clicks on the Central Concepts page of the Notebook. A set of instructions are displayed on the Interaction Window.



APPENDIX C

Identifying Heuristic Associations

The second half of the third stage involves identifying of heuristic associations between concepts. By double clicking the middle mouse button (instructions are given in the Mouse Documentation Window). The Relationship Window (left front) displays existing relations for the concepts. Links can be added or deleted.

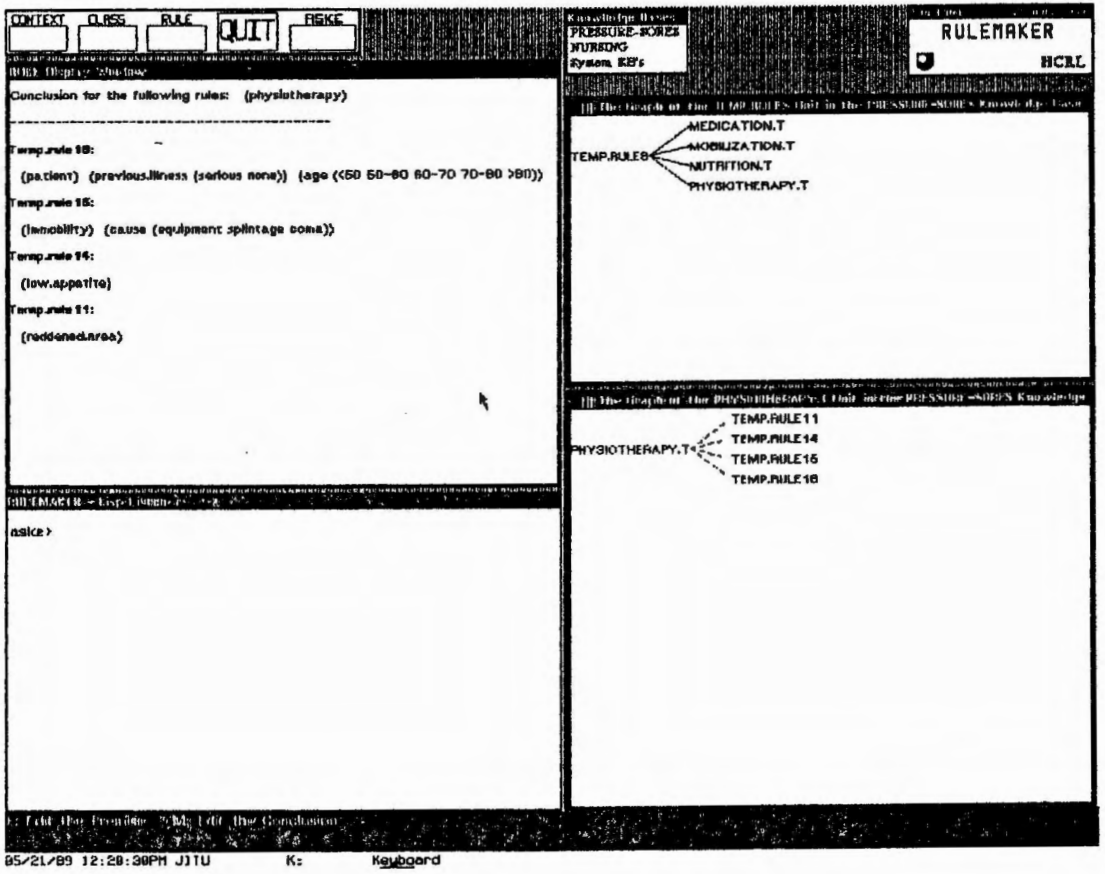


APPENDIX D

The RULEMAKER Interface

For the final stage, the interaction is transferred to the Rulemaker Interface. All the classes of if-then rules are displayed in top right window. The if-then rules are automatically generated from the heuristic associations. The condition part of the rule comes from the data concepts. The solution concepts make up the action part of the rules. Furthermore, all rule classes are named after the solution concepts.

The graph of a sub-class is displayed in bottom right window. In this window, the user can add, delete or merge rules. The rules are displayed in the Rule Display Window (top left).



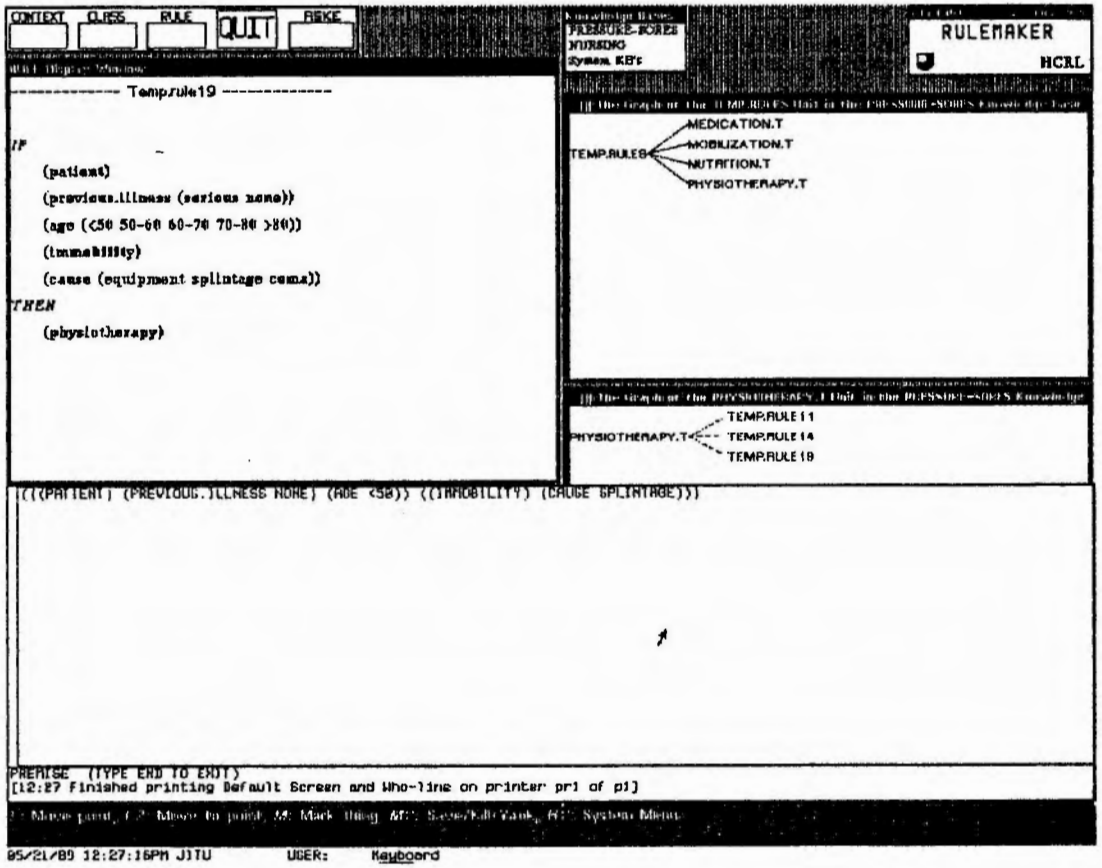
APPENDIX E

Creating Rules

Intermediate-level rules are produced from the if-then rules by merging and editing the condition and action part of the rules. For example, rules 15 and 18 are merged to produce rule 19. The editor window is opened by clicking the Rule Display Window. The edited version of the rule is as follows:

IF (patient) (previous.illness none)
(age <50)
(immobility) (cause splintage)
THEN (physiotherapy)

This rule will subsequently be translated into an intermediate-level.



A GRAPHICAL FRAMEWORK FOR KNOWLEDGE ACQUISITION AND REPRESENTATION

June 2nd, 1989

Jyrki Kontio
Pertti Lounamaa

Nokia Research Center
Knowledge Technology
P.O. Box 156
02101 ESPOO
Finland
tel: (+358 0) 43761
telefax: (+358 0) 455 2091

ABSTRACT

We present a framework that integrates knowledge acquisition and knowledge representation. The development of a system is carried out by concentrating on four main components of a knowledge-based system: architecture, domain object modelling, dependencies and control. The conceptual framework integrates the different components, allows different view-points to knowledge and supports structured design of the system thus decreasing the representational complexity of the system.

We have used the framework described here manually in several large projects and have implemented parts of the framework on a graphics workstation.

Keywords: knowledge acquisition, knowledge engineering, knowledge representation, graphics in knowledge acquisition

1. INTRODUCTION

The purpose of this paper is to describe a graphical framework that can be used to support knowledge acquisition and structured development of large rule-based systems. The framework has following characteristics:

- graphical representation of rule-based knowledge
- different views on rule-based knowledge on different levels of abstraction
- detail-level knowledge can be presented in different ways interchangeably
- graphical representations can be converted to production rules or procedural code
- support for the four main knowledge components relevant to most knowledge-based systems

Although we have implemented parts of the system on a computer (the Graphical Knowledge Editor) it cannot be referred to as an 'expert system shell'. Rather, we see it as a rule generator that produces production rules that can, for instance, be ported to commercial PC-based shells.

2. DIRECTIONS IN KNOWLEDGE ACQUISITION

2.1. Knowledge acquisition techniques and tools

Knowledge acquisition is a widely recognized problem in the development of knowledge-based applications. Various techniques and tools have been suggested to help knowledge acquisition. Techniques used in knowledge acquisition include different forms of interviews, variants of protocol analyses and several indirect methods like multi-dimensional scaling as mentioned by Olson and Rueter (1987) and techniques based on personal constructs theory (see Kelly 1955 and Boose 1985). Computer programs for some of these methods have been implemented and for our purposes they can be categorized as follows:

(1) Tools for text management. These tools refer to tools that allow the knowledge engineer to manage and store pieces of text in a dynamical and flexible manner. Typically these text items are either interviewing notes or protocols from speak-aloud sessions (protocol analysis). Tools that are used are either traditional text processing programs or, increasingly, hyper-text programs. Several hybrid knowledge acquisition tools or environments, such as KADS or KEATS, have very powerful hyper-text features in them as presented by Anjewierden (1987) and Motta (1988), respectively.

(2) Tools for structuring knowledge. These tools are used to help the expert and knowledge engineer define relations between objects or concepts in the domain. This helps in structuring unclear knowledge. Tools in this category include specialized conceptual analysis tools (e.g. Shaw and Gaines (1988) clustering programs, multi-dimensional scaling programs and even induction algorithms when used to prompt the expert for further clarification of the expertise (Olson and Rueter 1987).

(3) Tools for knowledge representation. The knowledge representation facilities in different shells can also be considered knowledge acquisition tools if they can be used to clarify knowledge already represented in a system to the expert. Naturally, one expects these facilities to be understandable by the expert to be useful. The graphical representation facilities in some shells are the most useful examples of this (e.g. KEE or Nexpert). The graphics are most often used for representing rule trees or frame or object hierarchies. While graphical representations of frames or objects are helpful in knowledge acquisition, rule trees are often too cryptic for the domain expert.

Naturally, there has been much interest in integrating knowledge acquisition tools with knowledge representation tools. The so called 'knowledge diagramming techniques' aim to ease the process of knowledge acquisition by automatically generating detail-level knowledge representation languages from graphs or diagrams as presented by Hillman (1988). Our framework facilitates this combination of acquisition and representation. Furthermore, we believe that interactive graphics can actually become a very essential part in the knowledge acquisition process.

2.2. Knowledge modelling vs. Knowledge acquisition

Knowledge acquisition is often seen as a process where the knowledge engineer is responsible for conceptualizing and structuring the knowledge. Likewise, the role of the expert is often thought to be a passive one: to express his knowledge in verbal form and the knowledge engineer takes his notes or protocols for further analysis on his own. Many of the tools and methods supporting knowledge acquisition are designed to help the knowledge engineer in his interviewing or coding task: protocol and interruption analyses leave the knowledge engineer to interpret the results, indirect methods assume that the expert cannot formulate the knowledge explicitly (see e.g. Olson and Rueter 1987 and Shaw and Gaines 1988).

The indirect approach is very efficient in situations where knowledge is so complex and unstructured that the expert cannot express it explicitly. However, we have noticed that just like the expert is very skillful in using his expertise he is often very good at finding structures and models in his knowledge intuitively or heuristically. Therefore we have tried to activate the expert in the knowledge acquisition and representation process. In fact, we consider the building of a knowledge-based application a process where both the expert and the knowledge engineer together model the expertise. When trying to find an appropriate term for this process we were reluctant to use neither of the terms knowledge acquisition nor knowledge engineering because both terms imply that the expert has a passive role in the process. Lacking a better term we a referring to the process of expert and knowledge engineer jointly modelling the expertise in an explicit form as joint knowledge engineering or joint knowledge modelling:

> Joint knowledge modelling is a process where the expert and the knowledge engineer together formulate the knowledge explicitly at all levels in the system, from the over-all structure to the detail level knowledge.

The feasibility of joint knowledge modelling is largely dependent on (1) the expert's ability to analyze and formalize his knowledge, (2) the clarity of the knowledge itself, (3) understandability (transparency) of the knowledge representation method used and (4) the time available for the project from the expert.

We would like to note that the term joint knowledge modelling does not refer to building an abstract model of the domain like Hayward (1987) and de Jong (1988). However, our framework does not exclude their approach either.

3. DESIGNING THE APPLICATION - Development Approaches

We have used three alternative development approaches in our knowledge-based systems development: top-down modelling, bottom-up composition and model-based development. These three approaches are typically used as alternative knowledge acquisition strategies during development. Different approaches are used in different phases of a application project according to the situation. The most important factors related to the selection of the approach are clarity of the sub-domain being considered, type of knowledge within the sub-domain, the expert's ability to conceptualize things and the efficiency communication between the expert and knowledge engineer.

The top-down modelling means that the application is defined and represented hierarchically from conceptual schemas to details as the work goes on. The approach is naturally very attractive when it is possible - any documented system is usually represented in a hierarchical fashion. In our experience human experts are well able to participate in top-down modelling providing that the results of the modelling process (the representation) is understandable to them. Actually, our framework will be presented in a top-down fashion in the next few pages.

Unfortunately, top-down modelling is seldom a realistic approach on its own. The experts find it often difficult to think on the conceptual level for longer periods of time. It is also difficult to keep experts motivated as it is not possible to demonstrate working prototypes until late in the process.

Quite often the expert is not sure of the over all structure of the problem in the beginning of the development project. It may therefore be necessary to start constructing the system from the detail level upwards, which we refer to as bottom-up

composition. The conceptual structure of the system is formed as the detail-level knowledge converges into structures.

Several knowledge acquisition tools are designed to support bottom-up oriented development. They offer tools for managing interviewing notes or protocols (e.g. KEATS and KADS, see Anjewierden 1987). Also, the indirect techniques (clustering techniques, multi-dimensional scaling) are very useful in bottom-up design (Olson and Rueter 1987).

The idea of model-based construction of knowledge-based systems was suggested by Chandrasekaran (1987). A 'generic information processing task' is one that can be defined and used as a basis for development for several applications that fit in the generic model. In principle, once such models have been defined and are available, the knowledge engineer could save a lot of time by recognizing what model is relevant using it as a building block. The process of knowledge modelling bocomes more like veryfying a suggested model for the domain.

In our framework we have found that generic models are most useful at the architecture level of an application. An architecture of a similar application can be used as a basis for knowleledge acquisition sessions. We have not been able to re-use any significant parts of code or actual detail-level knowledge. Practically all of the code has been re-written for the new application.

4. KNOWLEDGE COMPONENTS

We have identified four main components that can be addressed separately during the development of a knowledge-based application. Although the components are strongly inter-related, they form a natural basis for separation of the knowledge modelling task. Addressing these components separately makes it easier to understand the structure and operations of the application, reduces the internal complexity within each component and facilitates modular structure of the application. Layers in the modelling language proposed by Hayward (1987) and methodology by de Jong (1988) are similar to our definition of components.

We will describe the different components in our framework in the following. We will also outline some tools and techniques we have used at each phase. Figure 1 presents an overall picture of the components in the framework.

4.1. Architecture

By system architecture we refer to steps or, in case of larger tasks, sub-tasks that can be identified in the knowledge processing task or in the system that is being developed. Instead of creating a comprehensive model of the relevant domain concepts like in de Jong (1988), our architecture refers to hierarchically organized, decomposed tasks that represent the steps in the problem solving process.

By decomposing the problem into individual steps and by defining the function and output of each task it is possible to gain a better understanding of the system to be developed. The process of decomposing is an iterative one. During the first cycle there may be only a few steps that are identified. However, as the output for each is defined, the system requirements begin to emerge. For instance according to Hayward (1987) requirements definition should be a separate activity from knowledge acquisition.

The steps in a knowledge processing task are identified and represented with a structured diagram (e.g. flowchart or structured programming chart). We have been using several different techniques and in figure 2 there is an example of program structure presented with JSP method (e.g. Peters 1981). It is possible to use different hierarchy levels if it makes it easier to see the process as a whole. Each phase should correspond to the steps or phases that the expert uses in his or her problem solving process. Ideally, each phase should correspond to a knowledgebase. For each phase one needs to define the output from that phase.

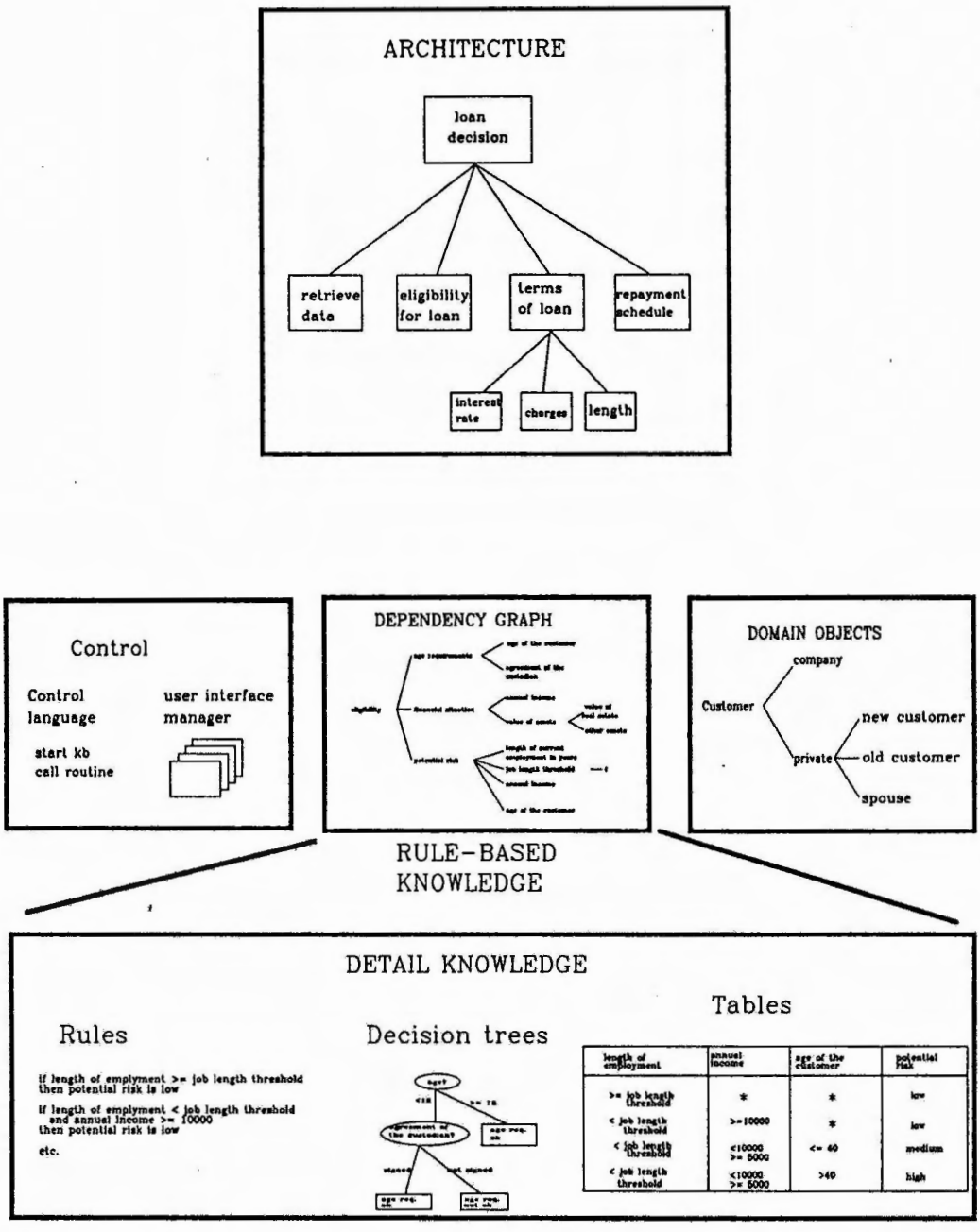


Figure 1: Components in the framework

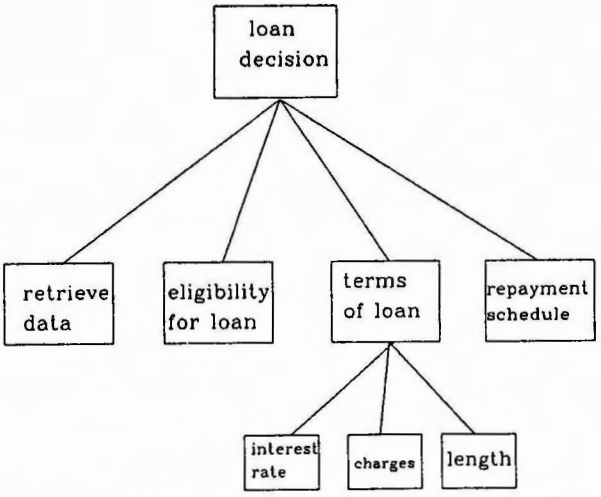


Figure 2: An example of steps in a task

4.2. Domain concepts

The concepts represent items or objects in the domain that are relevant to the process in question. In simple applications hierarchies and relations between concepts themselves are not always used within the application but in large and complex applications the analysis of concepts plays in important role.

If domain concepts are represented as objects in the system this makes the representation of knowledge more declarative, thus making the knowledge representation more powerful (Alberico 1987). Even when objects are not used in the implementation language, the analysis of domain concepts can help the expert and knowledge engineer to communicate. Often domain concept analysis results in a concept dictionary (data dictionary) for the project.

The analysis of domain concepts is carried out in iterative fashion. We perform the conceptual analysis by drawing an entity relationship map between the items that the expert recognises as important in the domain. This information is later used to help in the graphical representation of dependencies or in the design of frame hierarchies. We have not committed to any one format of entity relationship maps or conseptual graphs. Rather, we have kept the drawings simple and flexible and have adopted them to each experts' situation.

4.3. Dependencies (Logic)

The dependencies describe how concepts determine the values of other concepts. This is typically represented by production rules or equivalent. In our framework there are two different levels of knowledge representation for representing dependencies in the process. At a higher level, we draw dependency graphs (trees or networks). Each dependency graph corresponds to a step or sub-step in the procedural representation of the problem solving task (a sub-task in a JSP graph). At the lower level there are several alternative ways to represent the detail-level knowledge.

Dependency graphs are based on a hierarchical presentation of dependencies between concepts in the domain. A concept can be a real, physical or abstract phenomenon, or an internal, artificial concept that has been created during the knowledge acquisition process. A dependency tree consists of root node and child nodes that form a tree. Each tree begins with one parent node. A parent can have one or more children. Any of the children can, in turn, be parents to more nodes.

Let us consider an example. A subset of a sample rulebase about a loan decision knowledge is presented in appendix 1. The knowledge about eligibility for loan requirements is represented in figure 3. As the example shows, the dependency tree at this level does not contain any information about how the knowledge items effect the parent. They merely describe that there is a dependency.

Although the dependency tree is presented to the user as a tree, logically it is often a network. If a child has more than one parent it is presented as a child in several locations. A dependency tree becomes a network if dependencies for reoccurring items are drawn from the first occurrence. We have implemented a facility to view the dependency graph either as a tree or a network.

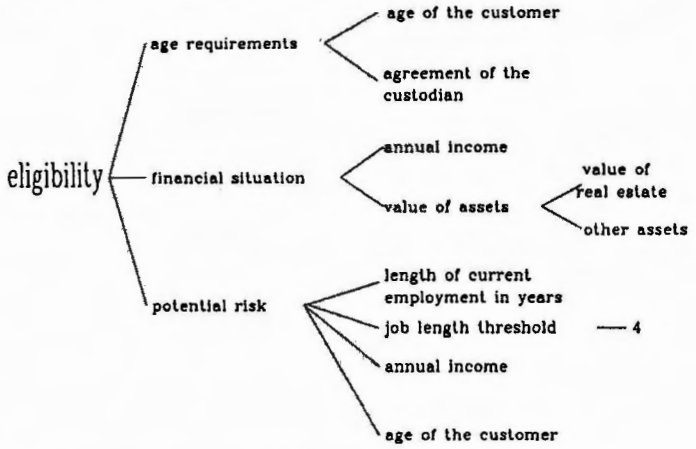


Figure 3: An example of a dependency graph

There are several ways to represent the detail-level knowledge about how children determine the values of their parents. We noted earlier that most experts present the abstract level of their knowledge in a standard, top-down form. For detail-level knowledge there are often different natural ways that experts use in describing their knowledge. Therefore, we have defined different alternatives for them. The techniques implemented so far in our graphical framework are:

- if-then rules,
- decision tables and
- decision trees.

Other representation alternatives that have not been implemented yet are (1) examples (induction, e.g. Quinlan 1986 and Thompson and Thompson 1986), (2) data in tables and (3) mathematical equations. The different forms of representing detail-level knowledge allow greater flexibility in knowledge representation (see e.g. Frost (1986), p. 45). The knowledge entered in one form into the system can be displayed in any other form to the user.

Detail-level knowledge is edited and entered into the system by activating a context, a group of items in the dependency graph. A context in a rulebase is defined as a group of knowledge items that have the same search criteria in the conclusion part of the rule. As we assume unfocussed search for the inference engine this means, in most cases, that one identifier (left hand side in an expression) in the conclusion part is the same in all knowledge items in the context. Stated otherwise, a context is a set of all child nodes of a parent in a dependency graph.

Once detail-level knowledge has been entered into the system, relations between rules can be presented as a rule tree or a rule network. The root node for such a tree is a query (or a node leading to alternative queries) that can be answered by the knowledgebase. All the rules that can answer to that query become children for the query. Each premise (condition in the IF-part) becomes a child for the rule.

4.4. Control

In expert systems literature control usually refers to different means of controlling the search of inference process (e.g. Vici, Mora and Sisto 1987). Although this is an important problem, the knowledge engineer often has to deal with more visible problem of managing processing between knowledgebases, user interface and other processing components of the system as well as performing iterations within the system.

One of the principles of our framework is that most of the control that is necessary in the application should be expressed at the same level where steps of problem solving task are represented. This reduces the amount of control that needs to be represented at the knowledgebase level which often is problematic in rule-based systems. Expressing control disturbs the normal flow of logic and frequent use of control structures results in an equivalent of 'spaghetti code syndrome' of traditional languages. Knowledgebases become difficult to understand and maintain and thus the efficiency of the application development decreases rapidly.

In the applications that we have developed, most common control aspects that cause problems in the implementation phase are (1) iteration control, (2) resetting inference, and (3) user interface requirements. Quite often these control procedures are embedded within the domain knowledge, thus increasing the complexity of the system. We believe that much of the control should be handled separately from the domain knowledge.

To make control issues manageable, we have separated the need for control into two different categories. Control language allows easy control of iterations, functional calls to knowledgebases and resetting of inference process. The most important language primitives are start knowledgebase, start routine, reset data, and do until.

The need to control the user interface is managed by a separate facility, the user interface manager. Priorities and links between display items (forms or questions) can be represented in the manager. During inferencing the user interface manager receives interface tasks from the knowledgebase. Questions and forms are presented to the user according priorities defined during the development.

5. IMPLEMENTATION AND USE

5.1. Implementation status

We have implemented a prototype of the graphical framework, called the Graphical Knowledge Editor, on a Sun workstation with Common Lisp. It can currently handle JSP-like structure diagrams, dependency graphs, if-then rules and decision tables. We are considering implementing them on a microcomputer during 1989.

Some other tools that we have used to support the framework include rule-based shells, hypertext software, JSP-charts, induction tools and decision network tools.

We have used the above described knowledge representation techniques manually in several large applications during the past two years, including: 3500-rule system developed for Okobank, 1800-rule system for SKOP-bank and 1000-rule system for Kansallis Banking Group (Kontio 1988a). Two of the systems handle and interpret mostly legal knowledge and one on them is a product configuration application containing bank's policies and subjective knowledge.

5.2. Use of framework

Our framework can be best used for applications that are currently developed with production rule techniques. We have used the framework with good results in the following application areas:

- interpreting the law, regulations or manuals
- loan evaluation
- analysis of customer potential
- straight-forward configuration problems
- product or service selection for a customer

The development of a knowledge-based application is typically an iterative process. Our framework allows the expert and knowledge engineer 'move around' the knowledge components and look at the application from different viewpoints. The framework itself does not impose any formal guidelines as to what steps should be taken at each phase. However, when possible, we suggest that the application should be developed according to top-down design principle.

To give a better understanding of our framework we have highlighted some of the differences to a successful knowledge acquisition tool (KEATS) presented in the literature (Anjewierden 1987 and Motta et al. 88).

Our framework is based on the idea of being open to different tools and techniques. This naturally allows a certain amount of flexibility, especially because we have built links to some tools that are often used (e.g. rule-based shell and induction tools). Therefore we do not offer any text management facilities except for a normal editor. Rather, we have used commercial hypertext packages for text management and have transformed such texts to our Graphical Knowledge Editor. The dependency graphs can also be used as a text management facility but, naturally, it is not as flexible as hypertext editors. KEATS has a very advanced hypertext-oriented editing facility and because it is a 'full' environment all components are well integrated with each other.

Both the KEATS system and the Graphical Knowledge Editor support interactive graphics. Our editor has a limited amount of representation alternatives in terms of specifying links between items. Although our approach is more limited in this sense, we feel that the dependency graphs provide a powerful structuring tool. Furthermore, using our framework does not exclude the use of other structuring techniques.

The KEATS system has a production rule interpreter. We have chosen not to implement an interpreter in our system. Instead, we are generating production rules that can be used by (currently by only one) commercial expert system shells. We have also developed a Rule Compiler that can convert production rules into procedural code of 'C' language (Kontio 1988b). While this gives substantial performance improvements it also makes rulebases very portable and embeddable.

Our framework does not exclude the use of other knowledge acquisition tools or expert system shells. Although we have implemented the Graphical Knowledge Editor on a computer, we have made it an open system. It is possible to read in rulebases developed with a commercial PC-based shell or some knowledge acquisition tool and display them graphically. It is also possible to generate the rulebases with our framework and execute them on a commercial shell. Actually, we are trying to develop our framework in such way that it could act as a center point between several different environments allowing transfers of different structures of knowledge between tools.

6. CONCLUSIONS

The usefulness of graphics for knowledge representation is supported by practical experience and strong intuition. Humans typically document complex information at least partially by using pictures, tables or figures. By using graphics one can conceptualise and compress large amounts of knowledge on a single screen thus allowing abstract views of the knowledge. Martin (1987) has presented that the use of graphics has the following main benefits:

- graphics enhance the communications between the expert and knowledge engineer
- graphics can contain much information in a very compact form
- graphics are often easy to understand

It is very difficult for humans to understand the function and operation of any large system without offering means to study the system at different levels of abstraction and from different angles. Our knowledge components and facilities for detail-level knowledge representation make it easier to understand the knowledge-based system being developed. The extensive use of graphics play an important part in this in our framework.

Our experiences indicate that the graphical framework presented here has the following benefits (see also Bohanec 1988):

- The three levels of abstraction enable the expert to understand and manage large knowledgebases.
- Knowledge can be represented in a form that is best understood by the expert.
- The expert can review and understand large 'chunks' of knowledge at a time making the development and maintenance of large knowledgebases easier.

- Detail-level knowledge can be presented in different ways helping the expert to review it by using different viewpoints to the same knowledge.

Our framework provides a uniform workbench for knowledge acquisition and representation. Our experiences lead us to suggest that it facilitates more efficient development and maintenance of knowledgebases, compared to production rule techniques alone.

The apparent power of pictures or graphs could be summarized by a modified saying:

One graph is worth a hundred rules

References

Alberico, Ralph (1987): More on Knowledge Representation, small computers in libraries, vol 7, no 11, dec 1987

Anjewierden, Anjo (1987): Knowledge Acquisition Tools, AICOM Vol. 0, No. 1, Aug 1987

Bohanec, Marko and Rajkovic, Vladislav (1988): Knowledge Acquisition and Explanation for Multi-attribute Decision Making, 8th International Workshop Expert Systems & Their Applications, Avignon 1988, conference proceedings, volume 1, Pp. 59-78.

Boose, J.H. (1985): A knowledge acquisition program for expert systems based on personal constructs psychology, International Journal of Man Machine Studies, 23, 1985

Chandrasekaran, B. (1987): Towards a Functional Architecture for Intelligence Based on Generic Information Processing Tasks, Proceedings of the 10th IJCAI, Milano, Italy; August 1987

Dionisi, A. Vici, Mora, M and Sisto, L. (1987): A tool for expressing control in expert systems, CSELT Technical reports - vol XV - no 6 October 1987

Frost, Richard A. (1986): Introduction to Knowledge Base Systems, Collins professional and technical books, 1986

Hillman, David (1988): Bridging Acquisition and Representation, AI Expert, November 1988

de Jong, Lieuwe Sytse (1988): Engineering of Expert Systems, Information and Software Technology, September 1988, Vol. 30, no 7

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

Kontio, Jyrki (1988a): Experiences from Development and Delivery of two large Expert Systems for Agricultural Loan Analysis, AI in Agriculture, food processing and agrochemical industries, Caen, France

Kontio, Jyrki & Lönnkvist, Olli (1988b): Compilation as an Alternative to Interpreting Inference Engine, Finnish Artificial Intelligence Symposium

Martin, James (1987): Recommended Diagramming Standards for Analysts and Programmers, Prentice Hall, 1987

Motta, Eisenstadt, Pitman & West (1988): Supporting Knowledge Acquisition in the Knowledge Engineer's assistant, Expert Systems, February 1988, vol 5, No 1

Olson, Judith Reitman and Rueter, Henry H. (1987): Extracting expertise from experts: Methods for knowledge acquisition, Expert Systems, August 1987, vol 4. No. 3

Peters, Lawrence J. (1981): Software Design: Methods and Techniques, Yourdon Inc. 1981,

Quinlan, J.R. (1986): Induction of Decision Trees, Machine Learning, vol. 1, no 1, 1986

Shaw, Mildred L. G. and Gaines, Brian R. (1988): Methodology for Recognizing Consensus, Correspondence, Conflict and Contrast in a Knowledge Acquisition System, paper presented at the Knowledge Acquisition for Knowledge-Based Systems , Nov 7-11 1988, Banff

Thompson, Beverly and Thompson, William (1986): Finding Rules in Data, BYTE, November 1986

SAMPLE RULEBASE

```
rule 1
  if  age requirements are acceptable
   and financial situation is normal or good
   and potential risk is likely to be low
  then eligibility is acceptable

rule 2
  if  age requirements are unacceptable
  then eligibility is unacceptable

rule 3
  if  financial situation is bad
  then eligibility is unacceptable

rule 4
  if  potential risk is likely to be medium
   and age requirements are acceptable
   and financial situation is normal or good
  then eligibility is subject to further analysis

rule 5
  if  age of the customer < 18
   and agreement of the custodian has been signed
  then age requirements are acceptable

rule 6
  if  age of the customer < 18
   and agreement of the custodian has not been signed
  then age requirements are unacceptable

rule 7
  if  age of the customer > = 18
  then age requirements are acceptable

rule 8
  if  annual income > = 20000
  then financial situation is good

rule 9
  if  annual income < 20000
   and annual income > = 10000
  then financial situation is normal

rule 10
  if  annual income < 10000
   and value of assets > = 50000
  then financial situation is normal

rule 11
  if  annual income < 10000
   and value of assets < 50000
  then financial situation is bad

rule 12
  if  length of current employment in years > = job length threshold
  then potential risk is likely to be low

rule 13
  if  length of current employment in years < job length threshold
   and annual income > = 10000
  then potential risk is likely to be low

rule 14
  if  length of current employment in years < job length threshold
   and annual income < 10000
   and annual income > = 5000
   and age of the customer > 40
  then potential risk is likely to be high

rule 15
  if  length of current employment in years < job length threshold
   and annual income < 10000
   and annual income > = 5000
   and age of the customer < = 40
  then potential risk is likely to be medium

rule 16
  if  length of current employment in years < job length threshold
   and annual income < 5000
  then potential risk is likely to be high

fact  job length threshold = 4

fact value of assets = value of real estate + other assets
```

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

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

ABSTRACT

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

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

1. DYNAMIC TENSION: TECHNOLOGY PUSH AND APPLICATION PULL

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

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

2. SEEDS: THE ARCHITECTURAL PROFILE SYSTEM

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

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

generalization were asked in an attempt to expand the constructs and link them into a network. Data were gathered from about 30 subjects who used the repertory grid elicitation program.

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

3. KAQ: A FRONT END FOR TEIRESIAS IN EMYCIN

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

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

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

1. How should the knowledge base be structured; in EMYCIN terms, how should the context tree be structured?
2. How should rules (or frames) be chained together to properly drive reasoning?
3. When uncertain knowledge is needed, how can certainty factors be controlled to improve knowledge base performance?
4. What are the variables (variously called attributes, traits, characteristics, parameters, concepts, and dimensions, or in EMYCIN, *PARMS*) that should be included in rules or frames? What are the potential values for these variables? Which ones are more important, and how does this affect the knowledge base structure and reasoning?

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

Boose thought repertory grids might be effective in helping Associates identify variables for rules. If an initial conclusion or solution set could be provided, then triadic elicitation could be used to help an Associate's expert generate an initial list of variables and values. Once these variables were identified, they could be used to help build rules in KS300. This use of repertory grids and, later, related methods, provided much of the technology push in 1983 and 1984.

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

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

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

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

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

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

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

Implications also proved useful for identifying knowledge at different levels of abstraction. They could, for example, point out "part of," "kind of," or subsumption relations. Implications could also show the relative independence of traits (the ENTAIL algorithm produced an entailment strength from 0.0 to 1.0). For some applications, it is important to make traits in rules as independent as possible; this technique provided a measure of trait independence.

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

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

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

- o Implication analysis (fuzzy set generalization)
  Show trait value relationships at higher level of abstraction
  Summarize grid information
  Note "problems" and fix manually (bad ratings, exceptions, consistency)
  Provide another view of data
  Measure trait independence
- o Implication Analysis and Examination
  Agreement
  Is it obvious?
  Is it new to the expert?
  Disagreement
  Are ratings correct?
  Is there an exception?
  Are traits being used consistently?

- o Hinkle's implication analysis
  Point out relationships, especially problems (ambiguities and equivalences)
- o DYAD dialog (incremental interviewing)
  Expand grid with important solutions and traits
  Use if expert has no initial set of solutions
- o N (neither pole applies) and B (both poles apply) ratings
  Note when a trait does not apply comfortably to a particular solution

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

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

4. ETS: EMBEDDED REASONING AND INTEGRATED ANALYSIS

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

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

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

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

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

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

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

Another rule could also be generated:

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

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

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

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

Our excitement about ETS was growing. A new type of knowledge acquisition tool had been built that helped people develop prototypes very quickly:

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

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

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

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

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

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

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

We had also produced a significant extension to repertory grid methodology:

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

- This was the first time repertory grids had been applied practically to knowledge-based systems; grid techniques had been successfully applied to other areas such as training and market analysis.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

PLANET had several useful analysis tools that Boose implemented in ETS. One of these was similarity analysis and review, where rows and columns in the grid were measured for similarity and difference. This was a powerful static analysis technique when applied to problem-solving information in ETS. For example, if two solutions were highly

similar (the ratings in their columns matched significantly), then the inference engine would not be able to discriminate between them very well. The review mechanism allowed the expert to add traits in a directed manner to help solve these problems. This same process, when applied to rows, could help find redundant or highly dependent traits.

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

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

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

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

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

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

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

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

5. NEOETS: HIERARCHICAL REPERTORY GRIDS AND MULTIPLE DATA TYPES

In January 1985, Boose taught a knowledge engineering course that was attended by a University of Washington psychology graduate student, Jeff Bradshaw. Already familiar with repertory grids, Bradshaw became interested in ETS and

started working at the Center in the spring. Bradshaw contributed key ideas to NeoETS in the areas of knowledge representation and reasoning.

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

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

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

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

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

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

Further developments of our workbench were aimed at handling knowledge base life cycle problems, the user interface, delivery of knowledge bases (and of the workbench itself), and integration with other tools and systems. This section will be expanded in the final paper to include specific needs that the features addressed and typical applications from this period.

1986

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

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

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

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

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

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

1987

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

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

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

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

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

1988

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

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

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

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

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

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

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

Aquinas and Other Knowledge Acquisition Tools

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

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

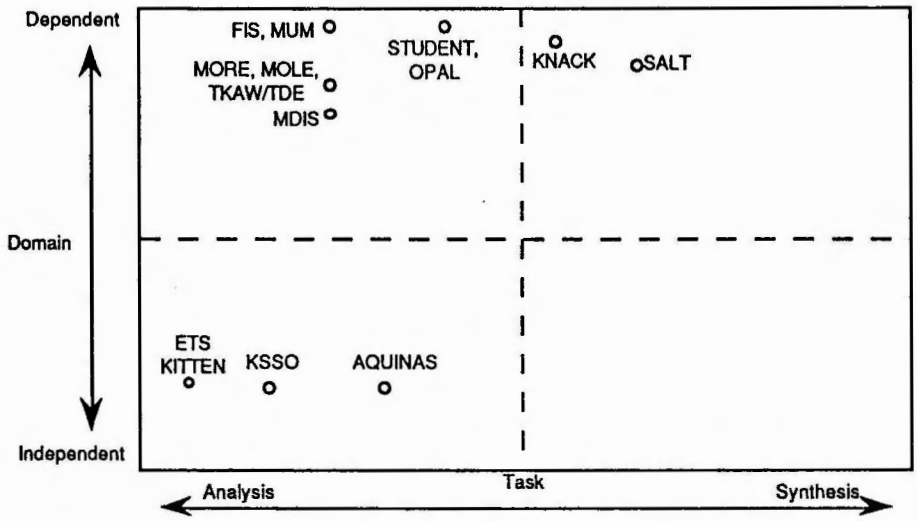


Figure 1. Knowledge acquisition tools plotted on two axes: *domain independence* and *problem type*.

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

Synthesis Problems

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

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

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

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

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

Life Cycle Support

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

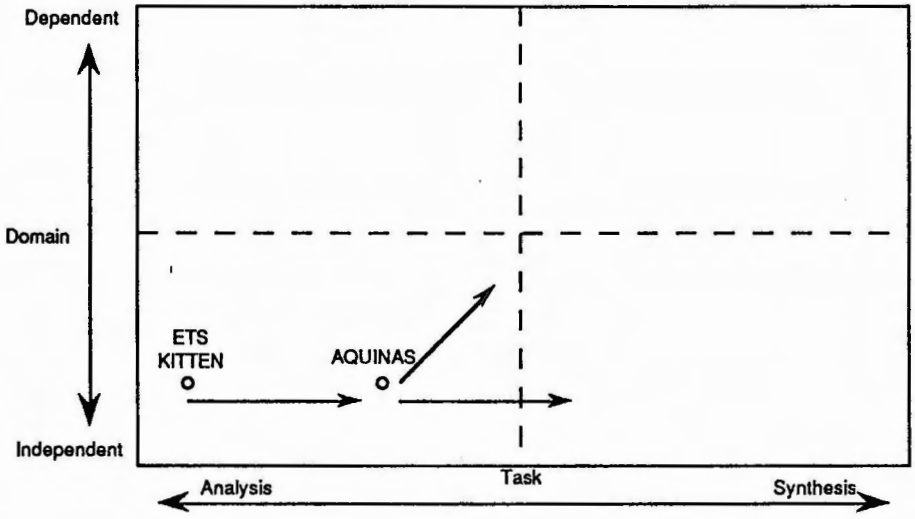


Figure 2. In the future we intend to expand Aquinas's knowledge acquisition power by embedding domain-specific knowledge and by augmenting Aquinas structures to handle synthesis problems.

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

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

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

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

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

8. CONCLUSION

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

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

ACKNOWLEDGMENTS

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

REFERENCES

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

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

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

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

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

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

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

Boose, J. H. (1988b). A Survey of Knowledge Acquisition Techniques and Tools, *Proceedings of the 3rd Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, November 1988, University of Calgary SRDG Publications, Department of Computer Science; also in *Knowledge Acquisition: An International Journal*, in press.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Kitto, C. M., and Boose, J. H. (1988). Selecting Knowledge Acquisition Tools and Strategies Based on Application Characteristics, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, in press.

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

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

Marcus, S. (1987). Taking Backtracking with a Grain of SALT, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 3, *International Journal of Man-Machine Studies*, Vol. 26, No. 4, pp. 383-398; also in Boose, J. H., and Gaines, B. R. (eds.), Knowledge-Based Systems, Vol. 2: Knowledge Acquisition Tools for Expert Systems, New York: Academic Press, 1988, pp. 211-226.

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

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

Shaw, M. L. G. (1980). On Becoming a Personal Scientist, London: Academic Press.

Shaw, M. L. G. (ed.), (1981). Recent Advances in Personal Construct Technology, New York: Academic Press.

Shema, D. B., and Boose, J. H. (1988). Refining Problem-Solving Knowledge in Repertory Grids Using a Consultation Mechanism, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, in press.

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

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

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

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

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

Centre MATRA-Espace de Toulouse
Phone: 61.39.65.45

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

Abstract :

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

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

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

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

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

1 INTRODUCTION

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

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

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

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

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

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

2 THE MACAO METHODOLOGY FOR KNOWLEDGE ACQUISITION

2.1 The main features of the expert cognitive model

2.1.1 Conceptual and cognitive models

When knowledge engineers build expert systems, their task consists in elicitating experts' knowledge, then structuring and formalizing it as reliably as possible. They aim at the target system being able to simulate the experts' problem solving behaviour. While they carry out this process,

the knowledge engineers form implicit or even empirical models of how the experts solve problems. For instance, many of them assume that all the expert's knowledge is available in his discourse during interviews, whatever the expertise may be.

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

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

Wielinga's models of expertise deal with a similar matter. Those models lay at different levels and describe the abstract structures that form the expert's domain knowledge. They differ from cognitive models because the expert takes part in their elaboration, and because the levels are suggested in order to facilitate the implementation. Unlike psychological models, they have not been experimentally tested previously[HAYWARD,87].

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

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

2.1.2 Some aspects of the expert model

The model we use studies the structure of knowledge through its evocation, storage and evolution [MICHEZ,86]. It assumes that, when an expert solves a problem, he achieves his knowledge by means of particular information selected in the environment: the objects of the expertise. These objects not only have physical characteristics, they also correspond to a mental model for the expert, labelled **functional unit**.

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

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

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

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

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

2.2 The MACAO method stages

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

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

1. **Initialization**
   This first stage is based on a sequence of dialogs between the expert and the knowledge engineer, combined with the observation of the expert at work. They specify the user's environment and the main features of the expertise. With the help of the repertory-grids technics, the expert classifies a selection of problems into categories. This set of information forms the software initialization nucleus.

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

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

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

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

3 KNOWLEDGE REPRESENTATION AS A SUPPORT TO STRUCTURATION

3.1 Knowledge representation: a paradigm to classify acquisition tools

In the existing knowledge acquisition tools, the problem of knowledge representation is tackled under two points of view. Either the process is oriented by the target development environment. Then, the elicitated pieces of expertise are represented directly in the knowledge base formalism. In spite of their powerful structuration, those systems rely on languages that may lack flexibility, limit or distort the expert expression. Or the tool has its own structures to store and analyse the knowledge. In this case, an ultimate translation is required to get the knowledge base.

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

3.1.1 Computational languages and acquisition tools

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

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

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

3.1.2 Visual representation

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

3.1.3 Mediating and intermediary representations

Many existing knowledge acquistion systems integrate several elicitation techniques which require a specific knowledge representation (cf KRITON [LINSTER,88]). Generally, these tools are de-

signed for the expert who gives his knowledge within a frame derived from an implicit model. The computational format is completely transparent for him.

Examples of such tools are ETS [BOOSE,86] or even AQUINAS [BOOSE,87], that expect the expert to describe the domain with objects and "constructs" (properties) from which inference rules are built up. Here, the knowledge representation offers a temporary means to facilitate the expert communication with the software, and the knowledge analysis.

On the contrary, tools such as KADS [HAYWARD,87] or MORE [KAHN,85] allow knowledge elicitation and representation in a more definite format. At the implementation level, the representation is transparent to the expert. He uses the items of an explicit domain model to enter his knowledge. The difficulty lies in the domain model definition. In the case of MORE, the model has been easily designed because the expertise dealt with medical diagnosis, which is quite easy to model.

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

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

Mediating representations are interesting supports to build a machine independent representation of the knowledge. These features meet most of the requirements that are specified for the MACAO representation in the next paragraph. For those reasons, the formalisms used in MACAO will be presented as mediating ones.

3.2 The knowledge representation specifications in MACAO

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

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

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

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

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

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

From those specifications, we have derived general principles to define the knowledge representation in MACAO.

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

In order to achieve the environment independance, we chose to use an intermediary representation. But as we wanted it to simultaneously follow the method prerequisites, a *ad hoc* **"mediating representation"** seemed to suit better. It is quite computer language independent in the sense that nothing inherent in the formalism forces a particular type of representation.

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

As those criteria present much incompatibility, we felt the need to distinguish an internal (and operational) representation from an external (and readable) one. Previously, we have assumed that the power of a tool includes the representation ability to take into account a part of the acquisition, mainly the structuration. In that case, it amounts to consider the capacity to connect the knowledge structuration and display, the internal and external representations. The use of a mediating representation subsumes that the knowledge can be easily translated into the target system formalism.

4 KNOWLEDGE REPRESENTATION AND EDITION IN MACAO

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

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

4.1 Internal representation

In the internal representation, we gave priority to readability, closeness to natural language and compatibility with a graphic edition, against complexity or operationality. As a consequence, we did not focus on the inference or inheritance mechanisms performances but rather on the achievement of the specifications of the representation.

To support the MACAO methodology, the Le_Lisp language seemed to be a judicious choice for its capacity for symbolic processing, indispensable in MACAO, its flexible use and the possibility

of its being extended. So, all the MACAO structures have been built from the Le_Lisp structured objects.

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

4.1.1 The basic environment

The AIDA environment of application development enables the construction of graphic interfaces for programs written in Le_Lisp . It offers elementary components (scrollbar, menus ...), elementary images (strings of characters, rectangles, icons ...) and some applications such as structured object editor, tree-editor, texteditor .... All these components can be used just as they are to construct complex applications, combined or improved for specific needs.

With AIDA, it has been possible to design a friendly interface, well adapted to MACAO. The dialog level suits the expert or the knowledge engineer according to the stage. The user is proposed various viewings, and may modify their configuration on the screen as needed.

4.1.2 The knowledge structures

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

The Le_Lisp objects As we want to give priority to the semantic ability, the chosen formalism focuses on objects. Le_Lisp offers structured objects and facilitates the creation of a hierarchy of structure types with the help of substructures. Structured objects are made up of several untyped named fields. Specific functions, called methods, can be attached to each type of object. Those functions check their parameters type and belong to the type package.

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

a name which identifies the functional unit,

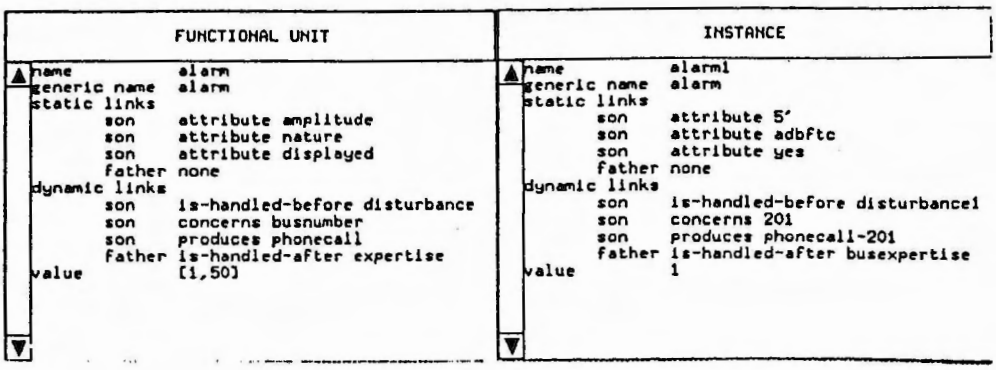
a generic name which is the same for every instance of the functional unit,

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

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

The functional units are clustered in three classes, according to their conceptual level: instances, objects and generic units.

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



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

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

The link connects two functional units. It has a name and a role which can be either an explanation text or a Le_Lisp function. Links are clustered in two major classes within a complete taxinomy:
- static links describe the world of the expertise: they connect an object to its different attributes, characteristics, properties or values. They express relations and structures located at the domain level as long as those relations are independent from the reasoning itself. Objects and concepts bound by such links form a kind of axiomatic structure that remains valid for all the problems;
- dynamic links describe the resolution process followed by the expert (a given object is calculated from, depends on, ... the ones it is linked to). They may represent the inferences that the expert develops when selecting or running a procedure.

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

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

The procedure is a logical sequence of functional units shared by several problems of a category.

4.1.3 Possible functions on the structures

For each structure, a set of functions is available through a specific menu. Updating functions for lists of words, like glossaries or lexicons, are working like the ones of a text-editor. Any structure can be created, modified, instanciated or deleted by using internal functions, written in Le_Lisp. These functions are handled transparently by the MACAO interface. Functional units can be linked, either by creating the link whith the help of the associated structure editor, or by drawing it on the graph.

4.2 External representation

4.2.1 Graphs definition

We believe that a graphic representation of the knowledge structure is a powerful device for the expert to control the formalized knowledge at each step. This representation facilitates the adaptation to the user's dialog and explanation levels in the man-machine interface. Furthermore, it provides a convenient tool for the expert's direct control of the knowledge consistency. To answer these requirements, we chose to represent the knowledge in oriented graphs, where the vertices are "functional units" and where the edges are dynamic or static links, depending on the kind of graph they belong to. A vertex is made up by two kinds of fields:

-those which belong to the object or the concept it represents

-those which are specific to a graph vertex such as the displayed name and the connections with the other vertices.

The layout algorithm has been selected respecting the following criteria [TAMASSIA,88]:

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

4.2.2 Graphs conceptual levels and semantics

The knowledge is represented by two kinds of diagrams: graphs on the problem level, where the vertices are the objects, and graphs on the conceptual level, where the vertices are concepts. According to the information to be displayed, the vertices are connected by static or dynamic links. The graphs at the lower level can be considered as instantiations of the conceptual ones. This hierarchy of diagrams turns the final knowledge representation into a superimposition of graphs. To homogenize this structuration, the vertices are called "functional units" in both graphs. These units are the actual items handled by the expert and the knowledge engineer.

4.2.3 Possible functions on graphs

The user can make any desired alteration on graphs, like adding a link between two functional units, removing one, or adding a functional unit. The updating appears on the graph with a particuliar drawing (bold font). These alterations will be effective on the internal representation only when the user will ask for their validation. The validation passes the updating commands on the concerned internal structures, and provokes a real modification of the external representation. The positions of the vertices are calculated again to display a new graph.

These modifications are all the easier as friendly tools enable the user to displace a vertex or the whole graph, to set a specific vertex in the middle of the image, to link two functional units ...

To check the reasoning path, the mechanism of knowledge processing consists in going through the graph, following the dynamic links and applying the methods associated to the determinants.

4.3 The representation support to the method main stages:

In order to stress how the representation supports the method, we present here the software role and some of the functionalities allowed on the representation at each stage. To make the software running more explicit, we are going to refer to an example derived from our experience with bus-network controllers.

In short, the operator's task is to oversee eight bus-lines the status of which can be displayed on a central computer screen. As the computer records data about the evolution of the network

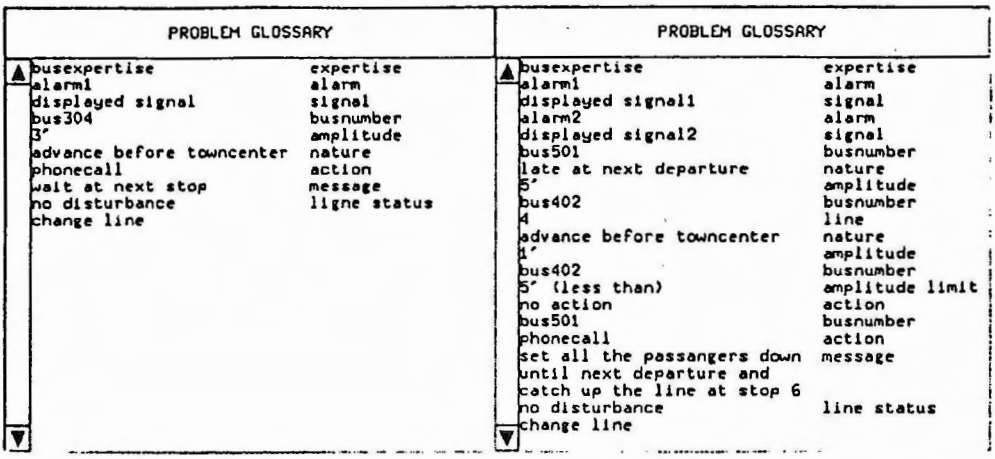
situation, a screen displays information such as the position of each bus on the line, bus deviation relating to the scheduled time...

4.3.1 A guide to conceptualize and formalize

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

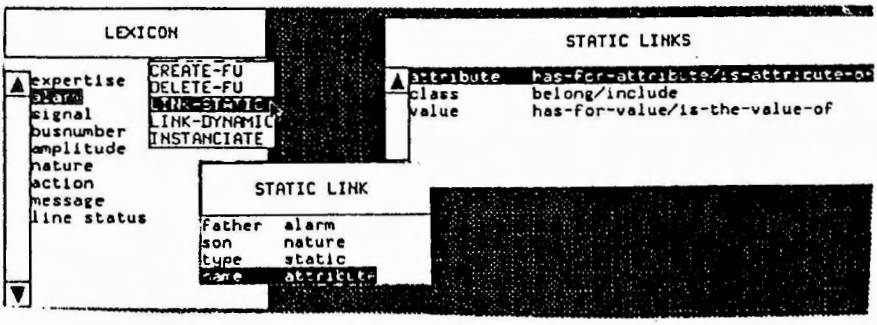
The left column of words, called the *problem glossary*, contains the names of the objects used for the corresponding problem. The expert writes them all along the solving, but the knowledge engineer may add some missing word after he has listened to the recorded explanations.

The right column gathers the generic names, picked in the domain lexicon, that are associated to the glossary words. They point out the concepts names that describe the problem solving at a more abstract level. As a matter of fact, this updating is the major part of the conceptualization.

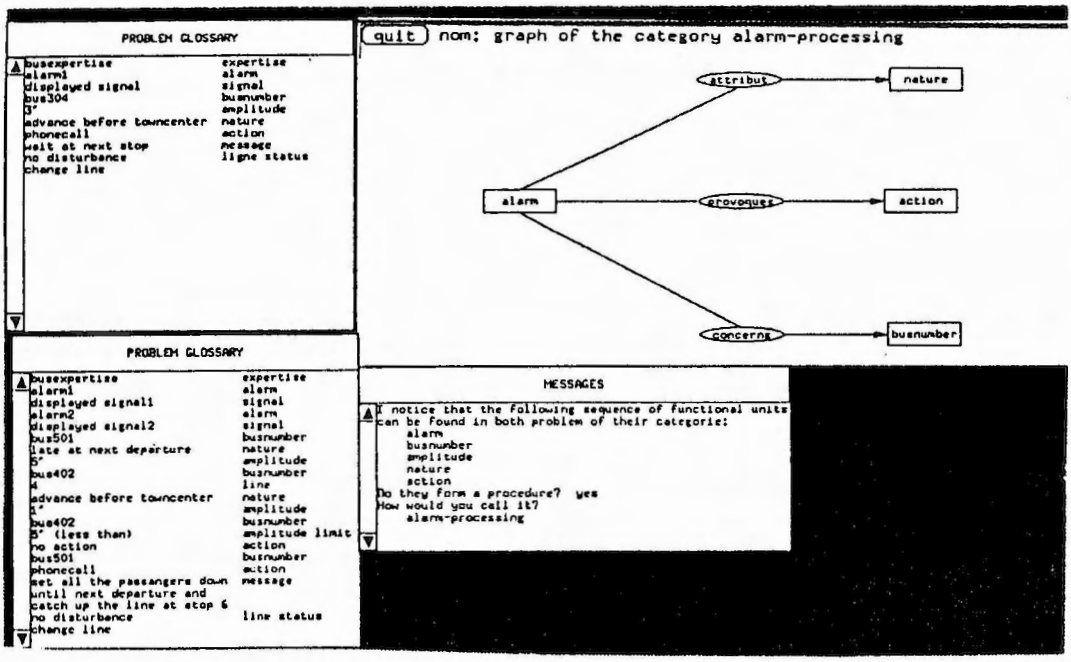


The knowledge engineer formalizes objects and concepts later. To this end, he creates a generic functional unit to formalize each lexicon word, an instanciated for the glossary items.

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



4.3.2 A support to the analysis



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

This example indicates how the software may suggest the knowledge engineer to identify a procedure. The system compares two problem glossaries until it finds out a common sequence of functional units. Then, it displays a warning message. The knowledge engineer can either validate and label the procedure, or ignore the message, according to whether he agrees or not.

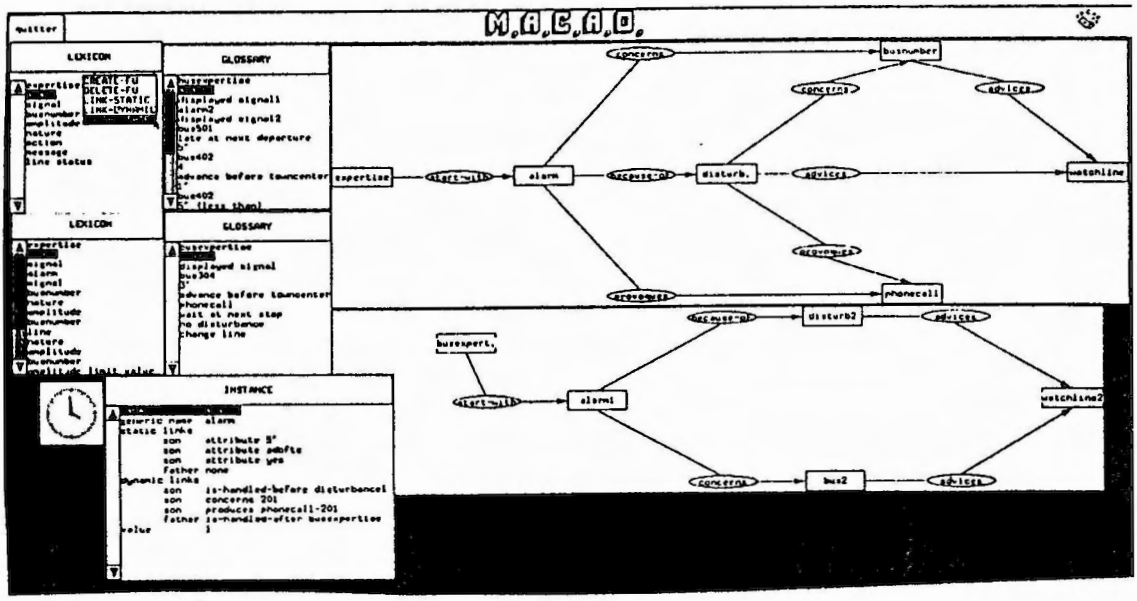
4.3.3 A means to ease validation

Beside the tool-box available at the conceptualization stage, the expert is proposed a set of functions in order to modify and validate the knowledge graphs. Those functions facilitate the graph validation process described earlier. The expert may go through the graph hierarchy from the global level to the problem graphs, until he detects the reasons for an error or a wrong structuration. On a static graph, the expert may check the static structure of any functional unit, control or modify the static links.

He is also given the possibility to edit and control the dynamic structure by adding, modifying or removing dynamic links on dynamic graphs. Moreover, the methods associated to the determinants can be tested, run for different set of values. The name and components of procedures can be controlled as well.

This "manual" validation is combined with an automatic control processed by the software. For instance, the consequences of any updating in a problem graph are passed on the associated category graph. The software shows the part of the graph concerned with the change, suggests some modifications and memorizes the expert actual updating.

Here is an example of a graph updating, and its consequence on the associated structures.



5 DISCUSSION AND CONCLUSION

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

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

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

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

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

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

References

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

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

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

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

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

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

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

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

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

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

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

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

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

[MUSEN,88] MUSEN M., Conceptual models of interactive knowledge acquisition tools. *Poceedings of EKAW'88, GMD Studien Nr 143*, Bonn (RFA), June 1988. p 26.1-26.15

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

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

Distributed Knowledge Acquisition as a Model for Requirements Elicitation

STEVE M. EASTERBROOK

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

This position paper presents a model for requirements elicitation based on the Distributed Artificial Intelligence paradigm of co-operating agents. Although parallels can be drawn between requirements elicitation and knowledge acquisition for knowledge based systems, there are difficulties particular to the former which prevent tools developed for knowledge acquisition being applied. These difficulties arise from the sheer quantity of knowledge needed during the software engineering process, and the fact that the knowledge originates from diverse sources which are often in disagreement. The nature of the design process requires that these disagreements should not be eliminated as they occur, as they usually indicate the need for design decisions. The distributed model will allow such conflicts to be accommodated during the elicitation process, until a decision is appropriate. In this way, the order in which decisions are taken is determined by the analyst. Effectively the elicitation is broken into two stages: firstly the construction of a set of viewpoint agents to capture the perspectives of the people with whom the analyst must interact, followed by the integration of these separate descriptions of the world into a single specification. The integration stage can be seen as a co-operative process between agents, allowing DAI techniques to be applied. A blackboard system has been implemented which supports this process, using a commitment logic to represent the agents. The aim of this system is to provide an environment which allows the construction of a distributed knowledge base to explicitly capture the individual perspectives for use throughout the software lifecycle.

1 Introduction

Any support for the analyst during requirements analysis must provide some guidance for the initial knowledge elicitation stage. This paper draws some analogies between requirements elicitation and the knowledge acquisition problem in AI. It then discusses several factors which make requirements elicitation particularly difficult, to illustrate why knowledge acquisition tools have not been successfully applied to software engineering. It goes on to suggest that ideas drawn from Distributed Artificial Intelligence can be used to structure the process and overcome the problems of scale.

Requirements Analysis is generally taken to be that portion of the software engineering process that begins with an informal statement of need and produces a formal specification. Ideally, this latter document should contain all the information that might be needed during the design stage, although in practice, the clients perceived needs will change during the lifecycle of the system. If the requirements aren't captured adequately, the result is uncertainty throughout the software life-cycle.

Current support for analysis is generally limited to providing representations for the knowledge gathered, with little or no guidance on how to acquire the knowledge [Kramer, Ng & Potts 1987]. A more useful form of support would be to allow the analyst to build a knowledge base which can be consulted throughout the lifecycle of the software. This would avoid the situation where anything left unsaid by a formal specification document remains in the analyst's mind.

1.1 Requirements Elicitation as a Knowledge Acquisition Task

Requirements Analysis proceeds by building a description, to understand how the task is currently accomplished, and how it might be improved, in order to specify exactly what the automated system must do. This process is naturally knowledge-intensive, and is essentially a knowledge acquisition task.

The process of specification construction is an evolutionary one. The incremental steps either add more detail, or clarify existing parts by introducing exceptional case behaviour or retracting incorrect or over-simplified statements [Feather 1987]. Like knowledge engineering, this is based on the extraction and representation of information through some form of interaction with the experts

[Gaines, 1987], in this case the clients. Both have the same set of techniques available for extracting the information, including various types of interview, observing the "expert" in action (and subsequent debriefing), tutoring, and case analysis. Most of the information gathered in this way is needed throughout the lifecycle of the software.

In a large scale project, the information needed includes knowledge about software engineering methodologies and the target machine's architecture, as well as application domain knowledge. Furthermore, for maintenance purposes, knowledge about the target software and the design history are also needed. In his excellent survey of the use of AI in software engineering, Barstow [1987] describes the roles played by these types of knowledge and argues that AI techniques could greatly assist with the handling of this knowledge. The analyst is an expert in some of these areas (e.g. about software engineering methodology, and machine architecture), and a novice in others (most often, the application domain). Hence he needs to acquire knowledge from other people, and blend it with his own, to make recommendations for the specification. Even where the analyst has experience of the domain in question, his knowledge will be of a subtly different flavour to that of the client, being concerned with how the task can be automated. He will still need to consult the client about the detailed requirements of the particular situation.

1.2 Difficulties Presented by Requirements Elicitation

There are, however, a number of difficulties with requirements elicitation which confute the natural supposition that knowledge engineering techniques are applicable. These difficulties are well known in the software engineering community, (see Cunningham *et al*, [1985] and Finkelstein & Potts, [1985]) and case studies have shown the range of skills employed by analysts to overcome them (e.g. [Fickas, Collins & Olivier, 1987], [Adelson & Soloway, 1985]).

A typical expert system represents a small domain, which can usually be adequately represented by up to a few hundred rules. There is usually a single type of knowledge involved, namely domain knowledge relevant to the problem, and this can be represented in a uniform way. Where a single representation scheme does not suffice, it is rare to find more than two or three in the same system, due to problems of mapping between them (cf. the ADVISE system [Michalski & Baskin, 1983]).

Most importantly, the knowledge used to build expert systems normally originates from a single person, a captive expert. Whilst this does not guarantee self-consistency, it usually delivers it, and the expert will often make the effort to appear consistent, by helping to build rules which deal with apparent inconsistencies [Davis, 1979]. An important point to note is that, with regard to the resulting expert system, the captive expert is a disinterested party. Whilst the process of knowledge engineering may be of great interest to him, it is rare that he will ever seriously use the final system, and so he will be more concerned with how his knowledge is used rather than how the system is designed.

In contrast to this characterization of knowledge engineering as a working with a single disinterested source, requirements elicitation involves many disparate sources, nearly all of whom have to work with the resulting system, and so are most concerned to influence its design. This inevitably leads to a great deal of conflict, and it becomes hard to disentangle the requirements from the goals and perspectives (and associated opinions and biases) of the people involved. The need to choose between the diversity of methodologies available for software engineering strengthens the role of institutional policy and individual preference in the decision process.

This problem is compounded by the huge amount of information needed. Whilst consistency checking can be handled in a small knowledge base, as the size increases such checking rapidly succumbs to the combinatorial explosion. When a huge amount of information is involved any consistency checking is prohibitively difficult. In requirements elicitation, inconsistencies occur frequently, usually indicating a conflict between the interested parties. In such cases, the conflict represents the need for an explicit decision by the analyst, which should not be taken until all the appropriate information has been gathered. Hence all alternatives must be captured and accommodated to allow the analyst to delay these decisions as necessary.

As has already been mentioned, there are a great variety of knowledge types involved in the analysis process. The analyst will use many representations to capture this variety of information as a matter of course. These include dataflow diagrams, control-flow diagrams, entity-relationship models, data dictionaries, data structure diagrams, etc., each of which is intended to be used in

conjunction with some of the others. Indexing and cross-referencing these is a knowledge management task in itself.

All these factors contribute to the difficulties peculiar to requirements elicitation. Because of them, there is a paucity of computer support available to the analyst. Most existing software tools are geared to producing manual representations for the information. This means that most of the knowledge needed is held in the software practitioners' heads: only a small proportion is explicitly stored in the documentation. This increases problems of communication across the software team. If the analyst never explicitly states such knowledge, it can only be passed on by word of mouth to others involved with the software life-cycle.

2 Alternative Models for Knowledge Management

2.1 The Potential of Distributed Artificial Intelligence

Clearly, a model is needed that allows the analyst to acquire and manage a very large knowledge base. Such a knowledge base must be able to cope with conflicting information, and must allow the analyst to add to it without worrying about the integrity problems associated with scale. The field of distributed artificial intelligence (DAI) provides such a model: the co-operative agents paradigm.

The limitations of the centralist approach to Artificial Intelligence have inspired diverse approaches which question the basic assumptions of early AI work. The problem is especially apparent in Expert Systems technology [Sloman, 1985], [Bramer, 1985], where attempts to improve upon the generality and intelligence of existing expert systems by increasing the size of the knowledge base have met with an explosion in computational complexity. This has lead to renewed complaints that work in AI research does not scale up to real world problems.

Distributed Artificial Intelligence (DAI) questions the assumption that an intelligent program should consist of a single entity. Instead, it studies how intelligence can be modeled as the co-operation of a set of agents [Huhns, 1987]. Problem-solving activities are divided up among agents, according to their specialist knowledge, each capturing some logical part of the overall domain. As the system runs, the agents communicate partial solutions, and possibly control information (e.g. re-allocation of tasks), amongst themselves, until they converge on a final agreed solution. The premise is that intelligence is an emergent feature of co-operative behaviour.

The separate agents need not be consistent with each other, as the explicit communication acts control the sharing of information. Breadth of knowledge can be achieved by adding new knowledge sources rather than by expanding existing ones. The agents can then be kept to sizes which have been shown to work well, allowing them to maintain their internal consistency. In this way much larger knowledge bases can be built than has previously been possible, because the need to ensure global consistency, and all the effort that that entails, is removed.

Various degrees of sophistication are possible in the structure of the distributed model. The blackboard model is probably the simplest, as the separate knowledge sources need have no knowledge of each other [Erman & Lesser, 1975]. Removing the global blackboard requires the agents to communicate directly. In the BEINGS system [Lenat, 1975] they simply broadcast messages, usually in the form of requests for help, and hope that some other agent will reply. Other models insist that the agents should know exactly what other agents exist and be able to focus communication upon a particular target.

Early DAI work tended to assume that problems could be partitioned into totally independent sub-problems, and so co-operation reduced to trading tasks and sharing results. However, this is an unrealistic assumption, and a large number of problems cannot be partitioned in this way [Ginsberg, 1987]. Various techniques have been proposed to allow epistemic reasoning, especially in multi-agent planning systems. Agents must be able to reason about what other agents know, and are capable of, in order to make full use of their existence, and to co-operate effectively [Konolige & Nilsson, 1980]. Similarly, to communicate properly, agents must be sure their communications contain enough context to be understood, and that they will be useful to the recipient [Appelt, 1980].

Passing reference has often been made in the literature to the simplification of the knowledge acquisition process. For instance, Sridharan [1987] suggests that what he terms the capacity for

"Incremental Aggregation" is an important justification for the DAI approach, but has to admit that "no DAI framework exists with these properties of additivity". In general, it has been assumed that existing knowledge engineering techniques can be used to build the individual knowledge bases required.

As we investigated the applicability of the distributed agent model, it became clear that the model reflected the nature of the sources involved in requirements elicitation. Indeed, the blackboard system can be used as a metaphor for the elicitation process, with the analyst playing the role of a (self-scheduling) blackboard, which pulls in knowledge from many overlapping sources. Each knowledge source can represent some viewpoint or perspective, allowing the goals and policies associated with that perspective to be captured.

2.2 Frameworks for the Elicitation Process

Rather than a rigid formal process, the analyst needs a framework which can guide his or her expertise. Finkelstein [1989] lists the desirable features of such a framework, requiring that it be flexible, empirical (in that the model maps onto the results of observational studies), enactable, co-operative, and that it should be able to handle conflict.

Some recent methods, such as CORE [Systems Designers, 1985], do attempt to provide a prescriptive framework in which the analyst can apply his skills. CORE divides the analysis process into a number of stages, providing heuristics to guide each stage. These heuristics are useful because the method is intended to supplement the analyst's experience, rather than provide a fixed algorithm. The distinguishing feature of CORE is its use of *viewpoints* to give structure to the description. These represent the components of the system and its environment, and can be organizational, human, software, or hardware. In other words, every information processing entity with which the eventual system must interact, as well as those which will be partially or wholly subsumed by the system, are modeled as viewpoints.

However, the viewpoints used in CORE provide a structure only for process knowledge. Declarative knowledge about the domain, and about the history of the analysis process itself, is excluded from this structure. This arises from the view of analysis as a process of translating user intent into formal or semi-formal documents. The usual approach is to learn about the client, analyse the data, and then make suggestions. Presenting these suggestions to the client takes most of the time, as the client needs to be convinced that the analyst's solution will work [Fickas, Collins & Olivier, 1987]. A better approach is a more co-operative one, where the analyst and clients share their knowledge, and enter into a mutual process of making suggestions, and critiquing each other's suggestions. Unfortunately little computer support is available for synthesizing a solution from the various suggestions, nor for resolving the conflict inherent in the process.

Fickas [1987] has reviewed some of the most promising approaches for supporting this kind of co-operative process. These include Gradual Elaboration [Goldman, 1982], in which a small number of types of step are available to incrementally build the specification, Parallel Development [Feather, 1987], in which partial specifications are developed separately according to different development concerns, and then merged at a later stage, and Knowledge-Based Critiquing [Fickas, 1987] in which an intelligent model of the domain is used to debug a specification. Of these, the parallel development approach is the most successful at supporting co-operative work, as the merge process acts as a focus for conflict resolution, forcing the analyst to explicitly consider and document the interaction between different aspects of the specification. However, it is still not clear how the merge operations should best be carried out.

Backhouse [1988] takes a more extreme view of existing analysis techniques. He criticizes the preoccupation with procedure and information plumbing rather than the actual business activities saying that more attention should be paid to the agents involved in business and the social obligations entailed by their communication. He also criticizes the descriptions used (e.g. Entity-Attribute-Relation models) for being too arbitrary. He proposes a new approach which removes the assumption of a single object reality, and introduces the idea of business as a co-operative activity between agents.

The model described in the next section attempts to support this view of business as a co-operative activity between agents. By building a description of the agents, the analyst can focus on the interaction between them, and understand how the different views arise. This description can then

provide the basis for the analyst's own interactions with the people concerned during the process of specification construction.

3 The Distributed Knowledge Acquisition Approach

The approach to requirements elicitation proposed here is to use a set of agents to represent the views, or *perspectives*, of the people with whom the analyst must interact. The elicitation of knowledge will be separated from the integration, and techniques from DAI will be applied to the co-operative process of building the specification from the separate perspectives. The order in which the specification is built can then be controlled by the analyst, rather than by the order in which the information arrives.

The main aim of this approach is to provide support for the interactions between analyst and clients, and encourage them as explorations of the current state of the specification. It is vital that during this process the analyst should be in control: the method must only guide, rather than force, the order of discussion. Computerized support can assist this process in two main ways: documenting the information already gained, and guiding the discussions to areas which need more work.

3.1 Accommodating Inconsistencies

In order not to interrupt the interactions between analyst and client, the process of documenting the information gathered must not side-track the analyst into worrying about how the new information fits in. The analyst must be free to decide whether to work through the implications of a new piece of information now, or to leave this until a later stage. This means the knowledge base will frequently be in an inconsistent state. New information will be added which directly contradicts existing knowledge, or which is incompatible with existing assumptions. Any computerized support must be able to handle conflicting information, and ignore the inconsistencies until the appropriate time.

There is evidence to show that people can do this with ease. For instance, an analyst will quite often use a mental model of some process to reason about its behaviour, knowing full well that the model is an over-simplification which cannot handle certain cases. Similarly, a person will sometimes realise that their current assumptions are wrong whilst in the middle of a chain of thought, and simply ignore the problem until a later stage when they can think about it at leisure (Littman [1988] provides some examples of this behaviour).

Another factor in allowing the analyst full control of the process is to remove the need to consider the reliability of each new piece of information as it is gathered. In many cases the analyst will know whether what he is hearing can be regarded as undisputed fact, generally agreed principle, or contentious opinion. However, he will need to note the information no matter where it falls in this spectrum, and there will be cases where he doesn't know without further investigation. Basically, he must gain an overall picture of what each person thinks before being able to evaluate the various opinions.

Policies and preferences are problematic in requirements analysis as they cannot be represented as specific facts, nor as verifiable goals. Rather they are heuristic-like guidelines that restrict and channel the design process. They must be attributed to some agent in order to put them into context.

3.2 Using Agents to Represent Perspectives

The philosophy underlying the above comments is that the analyst must build up the whole picture before the most important decisions can be taken. This means that instead of maintaining a single "current version" of the specification, which is incrementally adjusted as new knowledge is acquired, the analyst needs to maintain a set of (often incompatible) perspectives which will not be merged into a single specification until a late stage in the process. This can only be possible if the analyst is provided with the tools to manage this increased body of knowledge.

The approach we are studying uses the notion of agents to represent the perspectives involved. Each agent is associated with a single source of knowledge, and that agent will always only contain information elicited from that source. Such agents will then remain available for inspection

throughout the lifecycle of the software, and so will enable anyone to discover how the perspectives influenced the specification.

Methods such as CORE (which use viewpoints), have already outlined the task of distinguishing the separate agents. However it is important to note the difference between the use of viewpoints in CORE and the use of perspectives here. When an analyst is building a description of a business activity, some form of structure must be imposed upon it. The notion of viewpoints is a logical development from processes and dataflows, which enables the processes to be more clearly identified with the agents that currently perform them. The viewpoints are used to structure a single, consistent description of the system, leaving no room for alternative descriptions. Everything the analyst is told must be fitted into the description by adjusting it where necessary.

Distributed knowledge acquisition, on the other hand, uses perspectives to provide alternative views to that imposed by a single description. The distribution is used not for structuring the specification, but for structuring the process of specification building. Each person can describe the system as they see it, rather than adding detail to someone else's description. In general, each person's description will be most detailed in the parts most relevant to them, and over-simplified elsewhere. However, it is important to be able to describe how points of interaction between agents appear from each perspective.

Each person has a specific idea about how their business activities proceed, what they want from the final system, and how it can help (or hinder) them. These individual goals colour their perspective of the requirements. Furthermore, organizations can be perceived as having goals, which may differ from those of any of its members, and will give rise to yet more perspectives.

Unfortunately, there is no one-to-one mapping between people and perspectives. This is because a person will often need to perform in more than one capacity, to carry out different parts of a job. Rather, the perspectives are associated with roles, whereby each role entails the notion of a responsibility. The responsibilities held by each person must be identified, so that they can be used as agents. An agent will then be used to represent the perspective associated with each responsibility.

Identifying the perspectives before the elicitation process begins is a difficult task, and the distinctions between the roles a person plays will often only become apparent during detailed discussions. This difficulty can be overcome by allowing the decomposition to evolve during the process. The analyst can begin with a set of agents based simply on people, or even groups of people, which can then be decomposed into roles as they are identified. This reflects the notion that the set of roles identified will depend on the sophistication of the description. For example, a first attempt at describing a library system might include the role *user*, which will be shared by many people. Later refinement may distinguish different types of user, such as *student user*, *staff user*, and *visitor*, with different needs and different goals.

3.3 Using DAI to Integrate the Agents

The process of knowledge acquisition never stops. Throughout the lifecycle of the system, the users' perspectives will change, and so will their needs. However, at some stage the analyst will feel he has acquired enough knowledge to begin to assemble the specification. At this point he will have a collection of agents representing the perspectives, each of which contains a partial description of the requirements. By avoiding the problem of integration up until this point, it would seem as though the analyst has saved up all the difficulties.

However, the integration of a set of well documented perspectives is a less daunting task than that of building a specification from scratch. Instead, the task is one of merging the separate descriptions into a single specification. The existence of these separate descriptions allows the analyst greater scope for exploration of the specification, without having to ask people about each idea. Several tools are available from DAI to assist with this process.

Firstly, the process is a co-operative one. The agents must act together, building upon each other's knowledge to construct the specification. Taking the blackboard model as an example, agents can add to the items on the blackboard to incrementally refine them, and combine existing items to produce new ones. In this model, agents are activated by the occurrence of relevant items on the blackboard. In this way, one agent can propose a small piece of the specification, which can then be refined by others.

The whole process is controlled by the analyst. In effect, he can act as a scheduler, deciding when and what an agent can add to the blackboard. He will monitor the blackboard, and can modify the specification being built or the scheduling used, to affect how the process proceeds. He can also stop the process at any point, to examine what the different perspectives have to say about some aspect, or to present the current state to the clients. Where the agents get stuck, for instance on a disagreement, or if some knowledge is lacking, they can refer to the analyst, providing as much information as possible, in order to make a decision, or to refer back to the clients.

4 An Example Problem

An small example can be used to illustrate the advantages of representing the perspectives separately, and delaying their integration. The problem below is analogous to that first described by Schoenmakers within a legal context, where the resulting faulty conclusion drawn by the judge was "the defendant is guilty of murder" [Schoenmakers, 1986].

4.1 The Problem

A software analyst interviews two employees of the client organization to establish the requirements (in this case for a library system). The first employee states that some fact **A** is true:

> We've got to have time limits for lending

whilst the second states that if **A** is true, then so is **B** (i.e. **A -> B**):

> If we've got to have time limits for lending
> then we'll need fines to enforce them!

The analyst puts these together, and concludes that **B** is true:

> In that case we'll need fines as a requirement

Unfortunately, both employees know that **B** is false (neither really wants fines!), as is consistent with their individual statements. The second employee stated the implication believing **A** to be false (she didn't want lending limits), quite possibly prompted by the analyst suggesting **A** because the first employee had mentioned it. The first employee, of course, wouldn't agree with the implication (unenforced limits can work okay).

Any doubt as to whether the problem is a valid one can be expelled by listening to clients talking about the requirements. Phases such as "Well, if **A** occurred we would need **B** to handle it", and "If we had **A**, we would want **B**" are often used in the counterfactual sense: the speaker still suggests the implication, knowing that **A** is not true. The rather simplistic scenario given above belies a tricky problem: it is more likely that the conclusion will be based on a long chain of inference, and the problem will go undetected until too late.

4.2 Where Traditional Methods Fail

This problem cannot be detected in a conventional knowledge base, as no information will be available to say that the two statements **A** and **A -> B** originate from different sources, and might not be compatible (Figure 1(i)). Even if the analyst found out both employees believe that **not B** is true, for instance by examining assumptions, and added it to the knowledge base, the problem would still remain, as the knowledge base is then inconsistent. Furthermore, the inconsistency is not a direct one and might only be detected by generating all the consequences of the current set of assertions, which is not usually practical (Figure 1(ii)).

Spotting the inconsistency is only half the problem. Assuming a good tool could help to detect the problem, there is no information available about which of the three propositions (**A**, **A->B**, and **not B**) are suspect. The analyst would need to decide which which position is right before proceeding, as no logic-based system can allow all three propositions to remain. This may force the analyst to make an early decision, in this case about lending limits, when such a decision may not yet be appropriate.

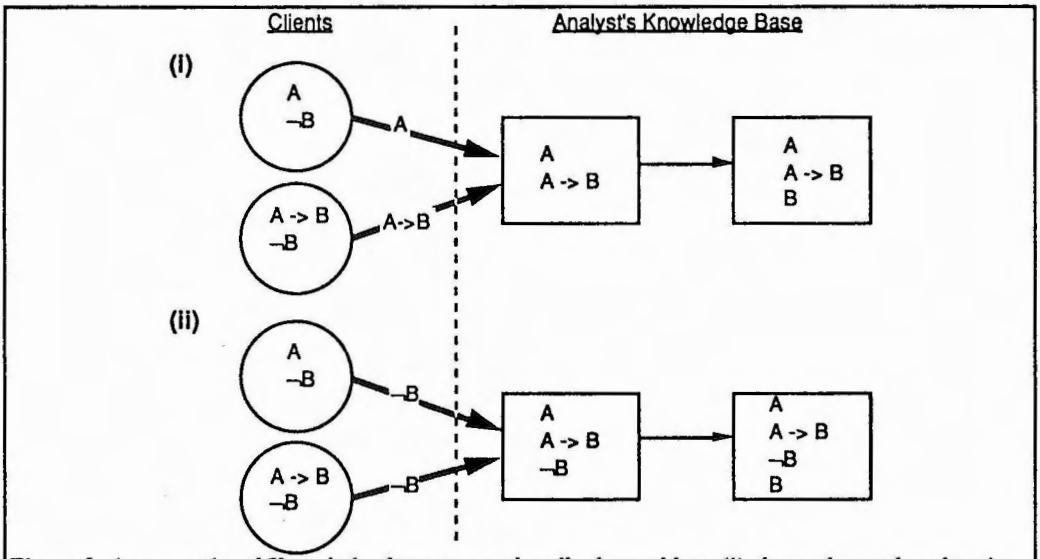


Figure 1: A conventional Knowledge base cannot handle the problem. (i) shows the analyst drawing the wrong conclusion based on the simple evidence of the employees' first statements. There is nothing to indicate to the analyst that the conclusion is wrong. (ii) shows an attempt to rectify the problem by asking the employees about 'B'. This causes further problems, as the knowledge base contains an inconsistency that can only be detected by generating consequences of the statements.

4.3 How the Distributed Approach Works

The distributed knowledge acquisition method is designed to provide a way to detect problems like this and to provide as much guidance as possible for their resolution. The first thing to note is that representing the employees statements as simple propositions discards important information. The fact that they came from different sources has been lost, as has all possibility of tracing who actually said them. In the Distributed Knowledge Acquisition method an agent will be associated with each source. (Figure 2(i)). Suppose the analyst then wishes to draw some conclusion about **B**. In this case he will ask the system whether **B** can be added to the specification.

While neither agent disagrees with the result **B**, neither can actually generate it, and so neither can actively support the analyst's conjecture. Only one agent has any information about **B**, and so the system indicates to the analyst that the person associated with this agent has stated **A->B**, but has said nothing about either **A** or **B** separately, and also notes that another person has said **A** is true. The analyst might now do one of several things, of which the most likely are:

To refer back to the people involved. In this case the system will make suggestions as to what to find out. For instance, the most fruitful line of enquiry would be to ask either employee for an opinion on **B**, or failing that, asking whether either accepts the other's assertion.

To allow the inference to be made on the current evidence. In this case, **B** will be included in the specification, but neither agent will add **B** to it's knowledge base, and each will note that it is being forced to agree to something it isn't sure about. These *notes* will lead to the agents persuading the analyst to verify this part of the specification with the people concerned at the first opportunity.

It is interesting to note what would happen in the case where the employees above had been one person acting in two different roles. For example, in a small departmental library which is run by some of the users, a staff member might suggest that lending limits are desirable while considering her role as a user, and then later reflect that lending limits could be enforced with fines when considering her role as librarian. In this case, the conclusion about fines can be generated for the specification, but will also generate a note for the agent to discuss **B** again later, because it has no actual commitment to fines. If the person is later asked about fines, and states that fines are undesirable, this will leave the agent in an inconsistent state. The *Analyser* system, described below,

handles such inconsistencies by splitting the agent, resulting in two sub-agents similar to those in figure 2(ii).

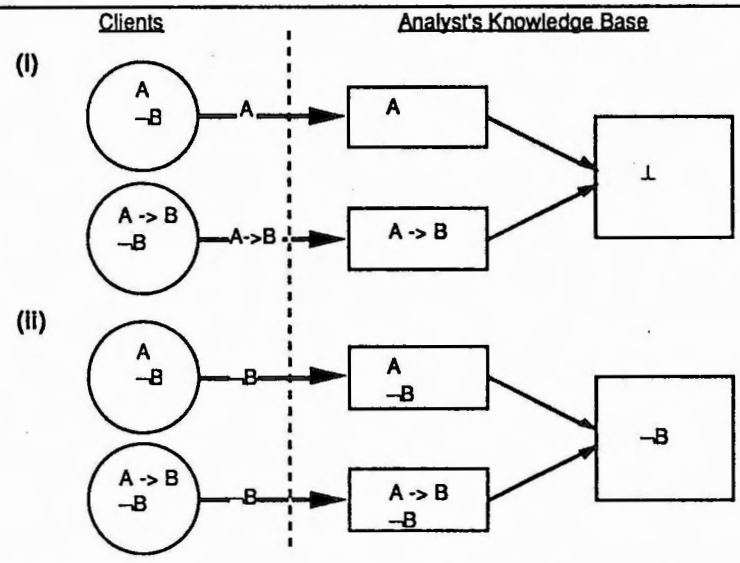


Figure 2: The distributed approach associates an agent with each employee, and will prevent any general conclusions until they are supported by both agents. In (i) there is nothing that both agents agree on. In (ii) the analyst has asked the employees about B, and as they both agree on not B, this can be placed in the specification.

5 The *Analyser* System

We are currently developing a system, called *Analyser*, based on the principles discussed above, in order to investigate the validity of the model. This section describes the architecture, and some of the functionality of the system. The program can be regarded as having two parts, a set of tools for knowledge gathering and a set of tools for specification building. This is not to say that the two sets of tools don't interact: indeed, the normal use would be to interleave the two tasks, using exploration of the specification to guide the acquisition process.

5.1 System Organization

Analyser is built around a blackboard system which provides the mechanism for interaction between agents. This is the basis for the tools provided for specification building, in which the analyst plays the role of scheduler for the blackboard. The agents themselves are created by the analyst to represent the sources of information used. A set of primitives corresponding to the speech acts used during elicitation (assert, deny, retract, etc.) are provided to record the discussions, by updating the agents.

Each agent is taken to represent only the assertions that a particular source is regarded as being committed to (by uttering them). This ensures that the agent is an accurate representation of the interactions between the source concerned and the analyst, and that no conclusions drawn by the analyst become part of the agent. *Analyser* uses a logic of commitment similar to that developed by Finkelstein and Fuks [1989]. This is based on a propositional logic, with the introduction of a third truth value, which represents the notion of *uncommitted*. Hence, an agent may agree with, disagree with, or be uncommitted about any assertion. The default value is uncommitted, representing assertions which have not yet been discussed, or are irrelevant to that perspective.

The analyst can speculate about what else the agent might agree to, using a scratch pad which is associated with each agent. These are not part of the agent, and so do not affect the contents of the agent. The scratch pads are used to hold any consequences derived from the current set of

commitments, as well as any assumptions the source appears to be using. Items in the scratch pad are not added to the agents set of commitments unless they have been discussed with the associated source. The main use of the scratch pad is for both the system and the analyst to create *notes*. These are used to indicate that further discussion is required about some item, in much the same way that Adelson and Soloway [1986] observed that designers make notes to remind themselves to come back to some item at a more appropriate time.

5.2 Knowledge Gathering

One of the major difficulties of any model which divides up knowledge using viewpoints is the problem of identifying the relevant viewpoints. As noted before, these do not correspond to the people involved, but rather represent roles. Not only is it not always clear which role a person is playing at any one time, but also roles can be associated with groups of people or organizations, making identification of the appropriate agent for an utterance difficult. For this reason *Analyser* is provided with some rules which enable it to recognize situations where a set of utterances may belong to two separate roles. This allow it to suggest the creation of separate agents to handle them.

The current version uses the simple rule that all the commitments made by a single role (and any consequences derived from them) should be self-consistent, and that if a contradiction arises within an agent or its scratch-pad, then either a mistake has been made, or the agent actually contains two roles. In either case, the analyst may not wish to examine the difficulty immediately, expecting the system to cope. When a new commitment is added to an agent, the system checks that it does not generate any contradictions. If it does, the system creates two sub-agents to handle the problem. The new commitment is placed in one sub-agent, and any previous commitments with which it conflicts are placed in the other. Any remaining commitments which are consistent with both remain in the original agent. The original agent still exists as far as the analyst is concerned, although any enquiries about what the agent is committed to will indicate that more than one alternative exists for some assertions. However, the agent can now be viewed as two separate agents which share some commitments.

At any time the analyst can examine what sub-agents have been created and why, in order to establish whether they are based on real distinctions between roles, or whether they are errors. Where a distinction between roles is identified these can be given meaningful names. Otherwise the analyst is free to resolve the conflict, or to leave it until later, in which case a *note* is generated by the system.

The process of refinement will eventually turn an original set of agents based on sources into a hierarchy of perspectives, where each inherits the contents of its ancestors. In particular, all the agents can be regarded as having an original global ancestor which holds any universal facts (including the rules of logic). These are automatically inherited by each agent unless the agent specifically denies them. This arrangement allows the analyst to start off with a simple set of agents based on the people with whom he interacts, which can then be refined as the process continues, to identify the actual roles.

5.3 Specification Building and Exploration

At any time the analyst can explore the current state of the knowledge base by experimenting with the specification. The specification is built incrementally by the process of co-operation using the blackboard. The analyst acts as the scheduler, enabling any agent to place an item on the blackboard for discussion. These items can then be amended or expanded by any other agents. When there are no further amendments to be made, the final result is considered to be part of the specification. Any agent can prevent an item becoming part of the specification by objecting to it while it is on the blackboard. The analyst then has the power to let the objection stand or fall by examining the relevant evidence (and pursuing further discussions with the relevant sources if necessary).

It is intended that the process of specification building will proceed before all the necessary information has been gathered, with feedback provided to guide the acquisition process. At present this is achieved using the *notes* facility. An agent will generate a note for itself if anything passes into the specification that the agent had no information on. This provides a safe-guard that prevents any perspective being missed out in some discussion areas. The notes will remind the analyst to raise the point with the agent's source. Similarly, an agent will generate a note if the analyst overrides it on the blackboard.

The specification is also considered to be an agent, and so participates in the co-operation process. This allows the system to moderate items on the blackboard according to what has already passed into the specification. This has the added benefit of alerting the analyst if items which conflict with the existing specification appear on the blackboard. Again, the analyst has the power to intervene at any point, by vetting the blackboard state.

There is also another agent maintained (invisibly) by the system which represents the analyst. Any direct alterations to the specification or the blackboard made by the analyst become commitments in the analyst agent. This also allows the analyst to make suppositions in order to explore the consequences. This agent simply records the actions taken by the analyst during the co-operation process, so that the analyst can examine and modify any assertions he has made. This will also warn the analyst if any intervention is inconsistent with previous interventions using the same mechanism as the agents.

As items pass from the blackboard to the specification, a record is made of which agent originated each item, and which agents contributed towards it. This allows the agents to build up areas of expertise within the specification. Any further query regarding that item can then be directed to the relevant source. Furthermore, if the agent subsequently retracts the commitments on which the item was based, the relevant part of the specification can be returned to the blackboard for renewed examination. These areas of expertise allow the system to give guidance on which source to ask for any missing information. If the analyst wants to establish whether an item should enter the specification, and there is insufficient evidence for the co-operation process to proceed, the system can identify relevant agents, and relevant questions in order to acquire the missing information.

Finally, it should be noted that as the specification is treated as an agent, it can be split into sub-agents if the analyst does allow it to become inconsistent. This allows the analyst to maintain alternative versions of the specification within the system, without having to discard any possibilities.

6 Conclusions and Future Work

This paper has outlined a programme of research into a distributed model to guide the analyst in the knowledge acquisition process. The most important aspect of this approach is that it supports the collection of multiple perspectives without forcing them into a single description. The framework provides a way to guide the analyst's skills, without unnecessarily restricting the interaction with the clients.

The model overcomes problems of scale by modularising the knowledge as it is collected. The structure suggested allows the knowledge base to contain inaccurate and conflicting information without the usual associated problems. Like the parallel development model [Feather, 1987], it provides a focus for the explicit merging of viewpoints, which is when most of the important decisions are taken. The key difference to the parallel development model is the provision of the DAI paradigm to handle the merge process.

It is important to stress that this is only a framework and does not provide a representation scheme for the knowledge used, nor for the specification itself. Several existing methods are possible candidates to fill this need. In particular, those which capture the notion of agents performing actions (e.g. [Maibaum, 1986], [Stamper, Althans & Backhouse, 1988]) would be most suitable.

6.1 Problems Encountered

We studied the Blackboard model as a basis for the integration process, due to it's simplicity. Unfortunately, the model has not proved to be powerful enough, in that the co-operation is restricted to agreeing or blocking other agents' suggestions. As there is no direct communication between the agents, they cannot reach any form of compromise through negotiation. The blackboard restricts how they can affect each others contributions. Part of the problem seems to be that the blackboard model assumes perfect co-operation - there can be no conflict between the agents. As has already been noted, conflict is inherent in the analysis process.

A further problem is that the blackboard model was initially designed for recognition problems where the process could be data-driven. Agents would be scheduled opportunistically by the appearance of relevant items on the blackboard. This is counter-productive in the DKA model, which

is intended to replace the restrictive data-driven aspects of the elicitation process with a model-driven approach. Accordingly, any further automation of the co-operation process can only be achieved by capturing the model the analyst is using, in terms of goals and operators. For any real-world problem, this will be necessary to free the analyst from having to deal with the details of the scheduling process.

There are a number of restrictions of the current version of Analyser, which it is envisioned will be lifted in later versions. Firstly, while the provision of the *notes* facility represents a powerful means to escape the linear ordering of the elicitation process, it is not yet clear how to decide when and how the notes are to be reactivated. At present the analyst can list the current outstanding notes, in the order in which they were generated. Clearly, there are other orderings possible, the most obvious criteria being importance and urgency. A simple importance ordering can be achieved by recording the circumstances in which the note was generated, so that, for example, an agent being overruled would be more important than simply a gap in an agent's knowledge. It would also be useful to allow the notes to be reactivated by events, in order to ensure that they did get re-examined. It is not yet clear how this triggering can be achieved.

The mechanisms for manipulating the agent hierarchy are also over-simplified at present. The only rule for recognizing when to split agents into separate roles is based on self-consistency. This assumes that role splits can be identified by direct contradictions, whereas often they will appear as different sets of goals which only give rise to conflict when they are implemented. The continued use of the agent model throughout the lifecycle of the system will allow the agent structure to be revised when the goal conflicts do surface. However, it would be helpful to use more sophisticated rules to decide when to split agents.

The agent structure can be compared to a decision-tree structure, in which the branches correspond to design decisions. Such a view may prove to be useful to the analyst in clarifying the important decisions of the design process. In order to exploit this similarity, tools must be provided to manipulate the tree, by re-ordering and regrouping the decisions. This could be achieved using a hypertext representation which allow multiple structures to be imposed on the same material [Conklin 1987].

6.2 Benefits of the DKA Approach

As well as allowing the analyst full control of the process of requirements elicitation, the distributed knowledge acquisition approach has other benefits:

No global consistency checks are needed - The knowledge base does not need to be in a consistent state to function. Each piece of knowledge exists within the context of an agent, and this context provides any extra information about the reliability and applicability of the knowledge. The need for techniques such as fuzzy logic, to cope with unreliable information is removed altogether.

The elicitation task can be shared - as each agent is constructed independently of the others, there is plenty of scope for parallelisation.

The model can cope with conflicting knowledge and alternative perspectives. Capturing these is vital to the requirements elicitation process, and is likely to be important in any domain where the problems are *wicked*. There is no correct description of a business system, only the views of the agents involved.

Opinions, policies and preferences can all be captured. In order to evaluate the constraints imposed on a software system, the goals and policies that produced those constraints must be represented. The analyst will then have better guidance in the decision making process. The blackboard system allow the agents to compare and integrate their goal structures [Hayes-Roth, 1985].

The model facilitates the recording of the decision making process. Decisions are taken only when the appropriate information has been gathered. The fact that this information, together with details of its origins, is already captured in the knowledge base, documentation of the process can be automatic. Proving support for the explicit integration of separate perspectives provides a focus for the decision making process. Also, the original agents will remain

available for examination, allowing the rationale behind the decisions to be seen.

Viewpoints can be added and refined throughout the lifecycle of the software. As already noted, the knowledge acquisition never ceases. The environment in which a system must operate will change, and these changes will be reflected in the perspectives of the people concerned with the system. The ability to examine what those perspectives were when the system was designed will allow more accurate pin-pointing of the changes that have occurred.

The process can be replayed. Capturing the decision making process, and the knowledge that guided it, allows that process to be replayed. This is an important benefit for software engineering, as backtracking on previous decisions is often necessitated by changing requirements. The traditional method of producing upgrades to account for deficiencies in the system can be replaced with a more natural evolutionary approach.

Whilst this study of the distributed knowledge acquisition model was prompted by the particular difficulties of the requirements elicitation task, there is no doubt that it is a useful technique for general knowledge acquisition tasks. Large knowledge bases must be structured in some way, and the DKA approach will allow them to be built incrementally, with that structure already provided. Recent research has suggested that intelligence, rather than being an individual attribute, is defined by the social system (e.g. [Gaines, 1987], [Miller & Drexler, 1988]). In this case, knowledge base technology will need to discard the idea of a single reality and embrace the notion of separate social realities. The distributed model provides a route towards this goal.

Acknowledgements

The author is indebted to Anthony Finkelstein, for patient and perspicacious supervision, and to Bharat Patel, for useful discussions on the ideas presented here. This work is being funded by the SERC, studentship number 87311891.

References

Adelson, B., and Soloway, E., 1985 "The Role of Domain Experience in Software Design", IEEE Transactions on Software Engineering, Vol SE-11, No 11, p1351-1360.

Adelson, B., and Soloway, E., 1986 "A Model of Software Design", International Journal of Intelligent Systems, Vol 1, p195-213.

Appelt, D. E., 1980 "A Planner for Reasoning about knowledge and action", Proceedings, First AAAI National Conference on Artificial Intelligence, p131-133.

Backhouse, J., 1988 "Simpler and Richer Cognitive Models for ISAD", Proceedings, IFIP WG 8.1 Workshop on Human Factors of Information Systems Analysis and Design, Imperial College, London, July 1988.

Barstow, D., 1987 "Artificial Intelligence and Software Engineering", Proceedings, Ninth International Conference on Software Engineering, p200.

Bramer, M. A., 1985 "Expert Systems, the Vision and the Reality", in Bramer (ed), "Research and Development in Expert Systems (Proceedings, 4th Technical Conference of the BCS specialist group on Expert Systems, 1984)", Cambridge University Press.

Conklin, J., 1987 "A Survey of Hypertext", Tech. Report No STP-356-86, Micro-electronics and Computer Technology Corporation (MCC), Austin, Texas.

Cunningham, R. J., Finkelstein, A., Goldsack, S., Maibaum, T., and Potts, C., 1985 "Formal Requirements Specification - The FOREST Project ", Proceedings, Third IEEE Workshop on Software Specification and Design.

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

Erman, L. D., and Lesser, V. R., 1975 "A Multi-Level Organization for Problem Solving using Many Diverse, Co-operating Sources of Knowledge", Proceedings, Fourth International Joint Conference on AI, p483-489.

Feather, M. S., 1987 "Constructing Specifications by Combining Parallel Elaborations", To Appear in IEEE Transactions on Software Engineering.

Fickas, S., 1987 "Automating the Specification Process", Technical Report No. CIS-TR-87-05, Dept of Computer and Information Science, University of Oregon, Eugene, OR.

Fickas, S., Collins, S., and Olivier, S., 1987 "Problem Acquisition in Software Analysis: A Preliminary Study", Technical Report No CIS-TR-87-04, Dept of Computer and Information Science, University of Oregon, Eugene, OR.

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

Finkelstein, A., and Potts, C., 1985 "Evaluation of Existing Requirements Extraction Strategies", FOREST Project Report R1, Dept of Computing, Imperial College of Science and Technology, 180 Queensgate, London SW7.

Gaines, B. R., 1987 "An Overview of Knowledge Acquisition and Transfer", in Gaines and Boose (eds) "Knowledge Acquisition for Knowledge Based Systems", Vol 1, Academic Press.

Ginsberg, M. L., 1987 "Decision Procedures", in Huhns (ed) "Distributed Artificial Intelligence", Morgan Kaufmann Publishers Inc, Los Altos CA..

Hayes-Roth, B., 1985 "A Blackboard Architecture for Control", Artificial Intelligence, No 26, p251-321.

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

Konolige, K., and Nilsson, N. J., 1980 "Multi-Agent Planning Systems", Proceedings, First AAAI National Conference on Artificial Intelligence, p138-142.

Kramer, J., Ng, K., and Potts, C., 1987 "Tool Support for Requirements Analysis", Tech. Report No DoC 87/3, Dept of Computing, Imperial College of Science and Technology, 180 Queensgate, London SW7.

Lenat, D. B., 1975 "Beings: Knowledge as Interacting Experts", Proceedings, Fourth International Joint Conference on AI, p126-133.

Littman, D. C., 1987 "Modeling Human Expertise in Knowledge Engineering: Some Preliminary Observations", in Gaines and Boose (eds) "Knowledge Acquisition for Knowledge Based Systems", Vol 1, Academic Press.

Maibaum, T. S. E., 1986 "A logic for the Formal Requirements Specification of Real-time / Embedded Systems", Alvey FOREST Project Deliverable Report No 3, GEC Research Labs, Marconi Research Centre, Great Baddow, Chelmsford.

Michalski, R. S., and Baskin, A. B., 1983 "Integrating Knowledge Representations and Learning Capabilities in an Expert System: The ADVISE system", Proceedings, Eighth International Joint Conference on AI, p256-258.

Miller, M. S., and Drexler, K. E., 1988 "Comparative Ecology: A Computational Perspective", in B. A. Huberman (ed), "The Ecology of Computation", North Holland.

Schoenmakers, W. J., 1986 "A problem in Knowledge Acquisition", SIGART Newsletter No 95, p56-57.

Sloman, A., 1985 "Why We Need Many Knowledge Representation Formalisms", in Bramer (ed), "Research and Development in Expert Systems (Proceedings, 4th Technical Conference of the BCS specialist group on Expert Systems, 1984)", Cambridge University Press.

Sridharan, N. S., 1987 "1986 Workshop on Distributed AI", The AI Magazine, Fall 1987, p75-85.

Stamper, R., Althans, K., and Backhouse, J., 1988 "MEASUR: Method for Eliciting, Analysing and Specifying User Requirements", Proceedings, IFIP WG 8.1 Conference on Computerized Assistance During the Information Systems Life Cycle, North Holland.

Systems Designers, 1985 "CORE: the Method", CORE manual issue 1.0, Systems Designers Scientific, Pembroke House, Camberley, Surrey, UK.

KNOWLEDGE ELICITATION FOR IMAGE INTERPRETATION

Maxine ESKENAZI[1] **Claudie FAURE**[2]

Simonetta FREDIANI[3] **Lorenza SAITTA**[4]

Keywords : Verbal Data, Thinking Aloud, Image Interpretation, Automatic Learning

SUMMARY

In this paper we present a methodology for acquiring and refining the domain-specific knowledge that an expert possesses and uses when interpreting visual data. The expert is asked to think aloud when analysing an image and to freely report in natural language. The verbal data are transformed into a set of example-dependent rules, this set is then automatically processed by a generalization algorithm resulting in a more compact set of domain-dependent rules. Arguments and results are given to show that the combination of verbal data recording and automatic learning ensures an easy and efficient way to elicit specific knowledge involved in image interpretation. The basic ideas behind the learning algorithm are described. The proposed methodology has been used for eliciting the acoustical and phonetic knowledge involved in speech spectrogram image segmentation. Results are presented and discussed.

[1] LIMSI - CNRS, BP 30, 91406 ORSAY cedex (France)

[2] ENST - CNRS, 46 Rue Barrault, 75634 - PARIS (France)

[3] C.E.N.S-CNR, Dip. di Automatica e Informatica, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 - TORINO (Italy)

[4] Dip. di Informatica, Università di Torino, Corso Svizzera 185, 10149 - TORINO (Italy)

1. INTRODUCTION

In this paper we present a methodology for acquiring and refining the domain specific knowledge that an expert possesses and uses when interpreting visual data. The data are speech spectrogram images and the interpretation process is based on the knowledge the experts have on the link between the production of speech and the resulting acoustic evidence.

When knowledge acquisition is performed, one of the most important problems arises from the fact that, usually, the expert cannot express what he/she knows in an organized and general form, he/she tends to give sparse fragments of knowledge, at different levels of generality and applicability. Stream of consciousness reporting is the most direct and widely-used method to gain information about domain knowledge. These reports are obtained by asking the expert to think aloud when performing image interpretation tasks. For each processed image, the verbal report is a set of data dependent arguments which are the trace of the domain knowledge activated on a particular example. The set of example-dependent rules will be automatically analyzed and transformed into a compact set of rules representing the domain knowledge, for the given data set.

The use of freely produced verbal data combined with a later refinement cycle allows both an adequate knowledge representation to be built and the acquisition task to be simplified. A good way to help the expert report the domain-dependent knowledge is to propose a correction task instead of a full interpretation task. Approximate interpretations, like those a novice expert can give, help the expert focus on errors and to verbalize the part of the knowledge relevant to his corrections. We use the results of automatic analysis of the first set of verbal reports to simulate the novice expert and to create conditions where the expert can correct this first knowledge base and therefore refine, update and complete it.

The automatic analysis is performed by a generalization algorithm, whose main characteristic is that it uses generalization operators, allowing the original set of rules to be grouped into a smaller equivalent one. This is achieved by introducing high level constructs, with increased representation power into the description language.

2. VERBAL DATA AND IMAGE ANALYSIS

Most of the Knowledge Based Pattern Recognition (KBPR) systems automatically solve the problem of visual data interpretation involving specific knowledge. These systems have to deal with (numerical) low level processing and (symbolic) high level knowledge. The complexity of this knowledge implies studies devoted to its acquisition, evaluation and representation. This results in knowledge base for the KBPR system.

The difficulties inherent in the process of Knowledge Acquisition (KA) from domain experts are well known and some approaches have been proposed in the literature to make this task easier, [BELKIN,87], [BOOSE,87], [KLINGER,87], [WIELINGA,87]. Nevertheless, no general methodology exists and none will be developped until the different categories of knowledge and the cognitive processes in which they are involved are not examined in detail. The expert's knowledge is by nature domain-dependent, in our case it is acoustic and phonetic knowledge. At a general level, the expert's task can be described as visual data interpretation involving specific knowledge.

The domain-dependent knowledge involved in image interpretation can be divided into the specific vocabulary and the rules. Each symbolic term of the vocabulary refers to a feature which can be recognized as a significant form on the image. The features are selected and then combined by a reasoning process that can be represented as a set of rules; image interpretation results from feature detection and rule application.

The expert reports his knowledge through verbal data. Despite the fact that information processing, which underlies the interpretation task, can not be completely elicited through introspection, there are strong arguments in favor of verbal reporting. The fact that experts can teach novices how they visually inspected and analyzed the spectrograms suggests that at least a

part of this knowledge is transferable and learnable by others. Moreover, we suspect that specific knowledge is not deeply encoded; most of the expert's knowledge has not reached a final stable state, therefore it remains accessible as far as modification and learning tasks are concerned.

Experimental results given in [ERICSSON,87] claim that verbal reports should not be discredited as a source of evidence about the reported cognitive processes. The limitations encountered (incomplete, inconsistent descriptions) can be explained by the nature of the main process (visual data interpretation) and the mutual interaction between the main process and the verbalization process. Perceptual routines which are used during interpretation are automatic processes, often activated in parallel. Also well-trained experts have "compiled" part of their knowledge - this changes a process under cognitive control into an automatic one. The more the expert uses automatic processes during interpretation, the less he will be able to report the intermediate stages of the information processing.

Another problem related to the task of image interpretation is that it is neccessary to recode non verbal into verbal representations. Verbal image description needs an analytical approach to represent the image as a set of local features and the relationships between them; this description can be very unnatural for the expert when the global appearance of the image constitutes the relevant information for its interpretation.

When the expert thinks aloud, there is mutual influence between the main task and the reporting, but the structure and the course of the ongoing cognitive process are not basically changed, and the information reported is valid knowledge. Verbal reports become more suspect when the expert is constrained to fixed representation schemes or strict ordering of questions. These structures no longer respect the natural framework of the expert .

In the case described in this paper, we are interested in the knowledge a phonetician uses to segment a speech spectrogram into phonemes or groups of phonemes. These images result from a frequency analysis of words or strings of words uttered by one speaker. The limit between two phonemes can be precisely located at one point on time (instant) or can be a certain duration. Different location candidates can also be present, only one will be chosen for the final segmentation. We asked the expert to interpret the image and to think aloud. The verbal reports were freely produced in natural language and recorded; they were then transcribed on paper.

3. KNOWLEDGE REPRESENTATION

In this section the elements of the knowledge representation scheme are introduced. First, the vocabulary, containing the terms used by the expert to denote acoustic-phonetic features and events will be described. Then, the logic language, used to formally represent these features and events, will be defined. The ***features***, considered by the expert at the acoustic-phonetic level can be summarized as follows:

Formants	:	F_1, F_2, F_3, F_4
Voicebar	:	High-Voicebar, Low-Voicebar, Voice Onset Time, Voice Termination Time
Silence	:	Silence in the Low Frequency range, Silence in the High Frequency range
Noise	:	Noise in the High Frequency range, Noise in the Medium Frequency range.

A necessary condition for segmentation is that some changes in the image structure must be perceived. These changes concern the features and are the ***events*** on which the expert bases her reasoning. An event is a local property of a feature (or a set of features) and its complete description is given by the property, the feature, and the time at which it happens. Only two basic properties have been necessary for our set of examples: ***Start*** and ***End***.

In order to automatically handle the information gathered from the expert, it is necessary to introduce a formalism to describe the features and the events. For this, a set $\mathcal{P}$ of predicates has been introduced. Each P(t) predicate has the following meaning:

P(t) = The event P occurred at time t

The predicates are organized into a generalization hierarchy, represented by means of a taxonomic tree. Examples of the defined predicates are reported in the following, as well as their meaning:

START(t) = The beginning of some feature is detected at time t
VTT(t) = Voice Termination Time is t
SFORM(t) = Some of the formants start at time t
SF1(t) = Formant F_1 starts at time t
SVOIC(t) = Voicing starts at time t.

Other properties reported by the expert, like *Increase* and *Decrease*, are taken into account by using the continuous-valued semantics associated to the representation language. In fact, the events may have a partial degree of truth μ (belonging to the interval [0,1]) associated to them, in order to represent their partial verification: for instance, the *Increase* of a feature at a given time can be represented as a *Start* with a value $\mu < 1$ associated to it.

In addition to the above-defined predicates, some others, describing relations among time instants, have been defined, permitting the representation of properties which relate a set of events (for example *Simultaneousness):*

SAME($t_1, t_2, \ldots, t_n$) = Time instants $t_1, t_2, \ldots, t_n$ coincide
BEFORE(t_1, t_2) = Time t_1 precedes time t_2

In order to handle combinations of events, the basic predicates belonging to $\mathcal{P}$ are combined in formulae of a logic language $\mathcal{L}$, derived from First Order Predicate Logic. Let φ be a well formed formula (wff) belonging to $\mathcal{L}$. Then φ can be recursively defined as follows:

- If $P(t) \in \mathcal{P}$, $\varphi = P(t)$ is a wff,
- If φ_1 and φ_2 are wffs, then $\psi = \varphi_1 \wedge \varphi_2$ and $\psi = \varphi_1 \vee \varphi_2$ are wffs,
- If φ is a wff, then ψ = q m [φ] is a wff. The quantifier q belongs to the set {ATL, ATM, EX} and m is an integer such that m≥1.
- If $\varphi_1, \varphi_2, \ldots, \varphi_r$ are wffs, then ψ = q m in {$\varphi_1, \varphi_2, \ldots, \varphi_r$} is a wff. The quantifier q belongs to the set {ATL, ATM, EX} and is an integer such that m≥1.

The intuitive meaning of the quantified expression ψ = ATL m [φ] is the following: given a known set $\mathcal{A}$, of tuples of constants, ψ is true iff there exist AT Least m tuples $a \in \mathcal{A}$ satisfying φ. Analogously, the quantified expression, ψ = ATL m in { $\varphi_1, \varphi_2, \ldots, \varphi_r$ }, is true iff AT Least m of the formulae $\varphi_1, \varphi_2, \ldots, \varphi_r$ are verified.

The meaning of the two other quantifiers can be deduced from the preceding one, according to the following relationships:

$$\text{ATM m } [\varphi] = \neg \text{ ATL (m+1) } [\varphi]$$
$$\text{EX m } [\varphi] = \text{ATL m } [\varphi] \wedge \text{ ATM m } [\varphi]$$

In this section only the syntax of the language $\mathcal{L}$ has been specified. The continuous-valued semantics associated to $\mathcal{L}$ will not be described here, for the sake of simplicity. In fact, the specific semantics chosen is not relevant and the one currently used may be found in [BERGADANO,88].

One last predicate completes the description language:

EX m [RUPT(t)] = "There are exactly m candidates for a boundary"

This predicate states that there are m time values sufficiently close to each other to be candidates for the same segmentation point or range.

Instances of formulae describing the examples are reported in the following:

2 = EX 1 [RUPT(t)] ∧ EF1(t) ∧ EF2(t) ∧ EF3(t) ∧ EF4(t)

5 = EX 1 [RUPT(t)] ∧ EF1(t) ∧ EF2(t) ∧ EF3(t) ∧ EBHF(t)

7 = EX 3 [RUPT(t)] ∧ EF3(t_1) ∧ EF4(t_1) ∧ EF1(t_2) ∧ EF2(t_2) ∧ VTT(t_3) ∧ SSIL(t_3) ∧ BEFORE(t_1, t_2) ∧ BEFORE(t_2, t_3)

15 = EX 2 [RUPT(t)] ∧ EF2(t_1) ∧ EF4(t_1) ∧ EF3(t_2) ∧ BEFORE (t_1, t_2)

22 = EX 1 [RUPT(t)] ∧ EBHF(t) ∧ SF1(t) ∧ SF2(t) ∧ SVOIC(t)

38 = EX 1 [RUPT(t)] ∧ DBMF(t) ∧ IBHF(t)

39 = EX 1 [RUPT(t)] ∧ SF1(t) ∧ SF2(t) ∧ SF3(t) ∧ SVOIC(t) ∧ EBHF(t)

41 = EX 2 [RUPT(t)] ∧ EBHF(t_1) ∧ EBMF(t_1) ∧ VOT(t_1) ∧ SF1(t_2) ∧ SF2(t_2) ∧ SF3(t_2) ∧ BEFORE (t_1, t_2)

The formulae of the $\mathcal{L}$ language result from the verbal report analysis of the expert. A frame structure, **BOUNDARY**, has aided this analysis. In it the natural language discourse of the expert was transcribed into a more structured version. For each segmentation example a frame consisting of four main sections is filled in.

The first section contains two slots, ***Left*** and ***Right***, filled with the interpretations (if any) given by the expert of the content of the left and right segments which are adjacent to the considered region of segmentation.

The second section contains the ***END*** and/or ***START*** point of the features; it is subdivided into three subsections, referring to the formants, to the features in the range of low frequencies and to other features, which may be identified anywhere over the whole frequency range. A number of ***Condition*** slots contain relations (=, ≈, ≤, <, ≥, >) between slot fillers (time of occurrence of events).

The third section contains information about features showing some degree of continuity across the possible segmentation zone. In particular, **Overlapping**, **Continuity**, **Increase** and **Decrease**.

The fourth section contains, in the ***RULES*** slot, the rules the expert mentioned to come to her decision in the specific case. Her decision is contained in the ***Zone*** and/or ***Boundary*** slots. If the expert was able to determine a precise boundary, the corresponding time instant is placed in the ***Boundary*** slot ; if not, the expert gives two boundaries, delimiting a segmentation zone by two points or a tolerance of some milliseconds around an instant. These two limits and/or this tolerance are put into the ***Zone*** slot. There may be more than one decision, when the expert considers more than one alternative to be possible.

For a complete description of the knowledge representation scheme for speech spectrogram segmentation, see [SAITTA,88].

4. RULE ACQUISITION

First the structure of the production rules, used to represent the knowledge acquired from the expert, will be described. For this, let us consider a set of alternative decisions (classes) and a set of rules, each one supporting one of the classes. Given a language $\mathcal{L}$, let φ be a formula belonging to $\mathcal{L}$ and let F be a set of examples. Then, EXT(φ,F) will denote the set of all the examples belonging to F and verifying φ.

The syntax of the production rules can now be defined as follows:

$$\varphi \xrightarrow{w} H \qquad (v) \qquad (1)$$

The right-hand side of (1) is a class, H, selected among the a-priori given set of alternatives. The parameters w and v are respectively the ***weight*** and the ***relevance*** of the rule, (see [BERGADANO,88]). Given a class H and a set of examples F, let E(H,F) be the subset of examples in F which are instances of H. Moreover, let C(H,F) be the subset of examples in F which are not instances of H. Then:

$$E(H,F) \cap C(H,F) = \varnothing$$

Given rule (1), we will say that a sample *f* verifies (is covered by) the rule iff *f* verifies the left-hand side φ. By using the preceding notions, the weight w of rule (1) can now be defined. Let F be a given set of examples, then:

$$w = 1 - \frac{|C(H,F) \cap EXT(\varphi,F)|}{|EXT(\varphi,F)|} \qquad (2)$$

In other words, w is the empirical estimate of the probability that a sample, verifying φ and belonging to F, is an instance of H. If w = 1, rule (1) does not make any classification errors. Notice that the definition of the weight depends upon the F set.

The relevance v of rule (1) may be defined as follows:

$$v = \frac{|EXT(\varphi,F)|}{|E(H,F)|} \qquad (3)$$

Also the parameter v is context-dependent; it is the empirical estimate of the probability that an instance of H, belonging to F, verifies φ. If v = 1, then φ is verified by all the examples of H, belonging to F.

As mentioned in Section 1, the process of rule acquisition from the expert can be performed in a cyclic sequence of steps. In particular, many specific rules, given by the expert at the beginning, are later generalized by means of an automated process. Some of the basic ideas which underly this learning system will be given here, whereas a complete description may be found in [FREDIANI,87].

4.1 - Generalization Algorithm

Let us consider a knowledge base consisting of a set of production rules following the syntax defined in (1). The left-hand sides (lhs) of the rules belong to a logical language $\mathcal{L} = \mathcal{L}(\mathcal{P},C,Q)$, where $\mathcal{P}$, C and Q are the sets of basic predicates, connectives and quantifiers, respectively. The problem addressed can then be stated in the following general terms: Given two languages, $\mathcal{L}_1$ and $\mathcal{L}_2$, such that $\mathcal{P}_1 = \mathcal{P}_2$, substitute another set of formulae belonging to $\mathcal{L}_2$ for the set of formulae belonging to $\mathcal{L}_1$, provided that a given equivalence criterion is satisfied. The two chosen languages $\mathcal{L}_1$ and $\mathcal{L}_2$ have their $\mathcal{P}$ sets dependent on the semantics of the domain of application, whereas the sets C and Q have the following definitions:

$C_1 = \{\wedge\}$ $C_2 = \{\wedge, \vee\}$

$Q_1 = \{\ \exists, \forall\}$ $Q_2 = \{\ \exists, \forall$, ATL k, ATM k, EX k$\}$

Basically, $\mathcal{L}_1$ contains only conjunction, whereas $\mathcal{L}_2$ may express conjunction, disjunction and numerical quantification.

The system which performs rule transformation is designed in such a way that all the algorithms used are domain-independent; in this way, when changing applications, the criteria that the system uses are the only items that need to be changed. Currently, the equivalence criterion states that the new rules should cover at least as many examples as and no more counterexamples than, the old rules.

During the process of rule substitution, alternative sets of new rules candidates are considered; user-defined criteria allow the system to choose the one which best satisfies these critreria.

4.2 - Rule acquisition in the first cycle

During the first cycle of KA, the expert describes the events occurring in one or many temporal locations for each segmentation. If the description concerns only one instant, then the decision is a precise boundary. If several instants are involved, then the decision can be a choice of one instant for precise segmentation, a zone of tolerance around one instant or a zone between two instants. Two classes of decisions have been defined, namely YES and NO. The YES class contains all those time instants which have been chosen as precise boundaries, whereas the NO class contains all the others. Then, φ formulae, belonging to $\mathcal{L}$, are left-hand-sides of rules supporting either the YES class or the NO class. Notice that an example of the NO class corresponds to a time instant which either was not chosen for a precise segmentation location (and was simply discarded) or became part of a zone. Only rules for the NO and YES classes have been extracted from the data, the decision about the choice between a precise boundary and a zone has been deferred to a higher level and is performed by means of a set of control rules, which predicate over rules of the first type, not over features.

A rule set $\mathcal{R}$ for YES and a rule set $\mathcal{S}$ for NO are inferred during the first learning phase. In order to learn the control rules, the cases of possible segmentation have been considered again; each case is now a single example, without distinction among the possible alternative boundaries. In particular, the cases have been grouped into three sets, according to the presence of one, two or three candidate boundaries. For each boundary t, the set of rules belonging to $\mathcal{R}$ and $\mathcal{S}$, activated by that boundary, have been used as a new description of the boundary itself; moreover, the initial description, involving the original predicates belonging to $\mathcal{P}$, has been kept as well. The learning algorithm first searches for control rules involving only the rules belonging to $\mathcal{R}$ and $\mathcal{S}$; if no satisfactory results can be achieved in this way, more complex control rules, involving both terms belonging to $\mathcal{R}$ and $\mathcal{S}$ and to $\mathcal{P}$, are searched for.

4.3 - Updating knowledge.

After the initial rules have been automatically processed, the transformed rules and their results were shown to the expert. Few strong contradictions were noticed between automatic and expert decisions. We call "strong contradiction" a case when no segmentation was automatically found on the examples or when there was a segmentation on counter-examples. Nevertheless, when more than one candidate boundary existed, the automatic decision did not fully agree with the expert. Differences arise when the control rules did not choose one boundary among a set of candidates (different locations were given as equally acceptable in the final decision), whereas the expert eliminated all but one candidate.

The confrontation of the automatic decision with the expert one was followed by a discussion on these differences. The major consequences were updating the decision classes definition and the meanings of some of the expert vocabulary terms.

The segmentation zones were the main source of problems. We did not discriminate among

the different possible origins of these decisions: they can be due to visual uncertainties in an event's location, uncertainties on the boundary choice when several boundary candidates are in competition or the unacceptable (or even refused) precise segmentation between two phonemes. Most of the zones are unprecise images where the precise location of the events is impossible, introducing meaningless refinements in the locations. The expert then decided to eliminate them by merging very close instants. This confrontation also showed that different mappings between the vocabulary terms and the image cues were used for image description. New predicates and new meanings of predicates are introduced in order to define precise conventions on the mapping.

The expert realized that the same results, or even better ones, could possibly have been obtained by simplifying the decision procedure, reducing it to a single step. The decision classes were redefined as **CHOICE** (precise boundary) and **ZONE** (between two precise instants), and the tolerance zone around a temporal location is eliminated; this is consistent with the above changes introduced in the predicates used for the descriptions.

4.4 - Second cycle.

The results of the second cycle are reported in this section. The formulae of the language $\mathcal{L}$, obtained after the first updating step, have been automatically processed for the newly-defined one-step decision and also for the previously defined two-step decision.

The two-step learning was performed on 42 examples for the **YES** class and 40 examples for the **NO** class in the first step and leads to the $\mathcal{R}$ and $\mathcal{S}$ rule sets which are used as predicates for the second learning step, in which rules for deciding between precise segmentation and zone are inferred. When the resulting rules were applied to the examples, there were 5 errors; in particular, 4 cases of precise segmentations where the expert considered it to be a zone and 1 case of zone, where the expert preferred precise segmentation.

More complete results are given for the new decision classes. The **CHOICE** class contains 42 examples and the **ZONE** class contains 12 examples. By applying the same algorithms as before, the following rules have been obtained:

Rules for the CHOICE class:

R_1 = ATL 3 in {SF1(t), SF2(t), SF3(t), SF4(t), SVOIC(t), SNHF(t)}
(w=0.83, v=0.45)

R_2 = ATL 3 in {EF1(t), EF2(t), EF3(t), EF4 (t), EVOIC(t), ENHF(t)}
(w=1, v=0.29)

R_3 = EX 1 [RUPT(t)] $\wedge$ [SVOIC(t) $\vee$ EVOIC(t)]
(w=0.91, v=0.45)

R_4 = EX 1 [RUPT(t)] $\wedge$ ENHF(t) $\wedge$ START(t)
(w=1, v=0.14)

Rules for the ZONE class:

S_1 = ATL 2 [RUPT(t)] $\wedge$ ATM 2 in {EF1(t_1), EF2(t_1), EF3(t_1), EF4(t_1), EVOIC(t_1)}$\wedge$
ATM 2 in {EF1(t_2), EF2(t_2), EF3(t_2), EF4(t_2),EVOIC(t_2)}
(w=0.903, v=0.755)

S_2 = ATL 2 [RUPT(t)] $\wedge$ ATM 2 in {SF1(t_1),SF2(t_1),SF3(t_1),SF4(t_1), SVOIC(t_1)}$\wedge$
ATM 2 in {SF1(t_2), SF2(t_2), SF3(t_2), SF4(t_2), SVOIC(t_2)}
(w=1, v=0.17)

For each rule, the weight w and the relevance v are reported.

When **CHOICE** rules and **ZONE** rules fire on the same case, control rules are applied in order to make the final decision. They are inferred as described in the first experiment. In the following, the $\mathfrak{R}$ and $\mathfrak{S}$ sets have the usual meaning:

$$\mathfrak{R} = \{ R_i \mid 1 \leq i \leq 4 \}$$
$$\mathfrak{S} = \{ S_k \mid 1 \leq k \leq 2 \}$$

Control Rules:

F_1 : EX n in $\mathfrak{R}$(t) $\wedge$ EX m in $\mathfrak{S}$(t) $\wedge$ GREATER(n, m) ---> Boundary in t

F_2 : EX n in $\mathfrak{R}$(t) $\wedge$ EX m in $\mathfrak{S}$(t) $\wedge$ [GREATER(m,n) $\vee$ EQUAL(m,n)] ---> Zone

F_3 : EX 0 in $\mathfrak{R}$(t) $\wedge$ EX 0 in $\mathfrak{S}$(t) ---> No Segmentation

The predicates GREATER(m,n) and EQUAL(m,n) are true iff the integer m is greater or equal to n, respectively. The intuitive meaning of, for example, rule F_2, can be explained as follows: if the number of rules supporting the YES class, which have been activated at a given instant t, does not strictly exceed the corresponding number for the NO class, then the segmentation is considered to be a case of zone.

By using the rules reported above, there was a total of 2 errors; specifically, 2 cases of precise segmentation where the expert wanted a zone.

5. DISCUSSION

Considering the results given by the two types of decisions, it is not possible to see a great superiority of one on the other.

The expert has reported almost exclusively positive examples: the transitions between phonemes were sure, the problem was to decide between a precise boundary and a segmentation zone. All of the knowledge involved in the segmentation task is that which allows the elimination of the locations where the events are not relevant and final decision making on the segmentation class.

The one-step decision is equivalent to the two-step decision for the two final classes (**CHOICE** and **ZONE**), but differs in the fact that it directly learns the final decision classes; this supposes that the decisions have to be made on positive examples. The **NO** rules of the two-step decision are explicit knowledge referring to negative examples. Part of the control rules are aimed at accepting or refusing a segmentation when **YES** and **NO** rules are in competition. The one-step decision concludes no segmentation when no rules can be applied. The lack of explicit knowledge to decide that a set of events is not relevant for segmentation seems to be an impoverishment as compared to a formal point of view.

A test has been conducted on new spectrograms in a pattern recognition task. This test has been performed by a novice human expert, who is able to recognize the features but has problems in discriminating relevant events and has poor performance in segmentation.The one-step and two-step sets of rules are activated anywhere that events are detected on the image. Events are perceived at different locations in the image, these instantes are candidate boundaries for segmentation (one or many instantes for each segmentation). Eight spectrograms corresponding to 47 segmentations have been described. The two sets of rules lead to a decision which is exactly the same as that of the expert for 29 segmentations. The remaining segmentations are divided into 4 sets: set A (resp. B) where only the two-step (resp. one-step) decisions are the same as the expert's (4 segmentations for A, 5 segmentations for B), set C where no rules lead to a decision

matching that of the expert's for 5 segmentations, set D where the decision obtained on 4 segmentations are acceptable by the expert who, nevertheless, preferred another.

An obvious result is that the beginning of a string of speech needs special rules: 3 out of 5 errors in C and 2 out of 5 errors in B correspond to this situation. The results also show that the two-step decision seems better. When the correct answer is given by only the two-step decision, the one-step set of rules gives a very erroneous answer (precise segmentation instead of no segmentation). On the contrary, and without considering the case of the beginning of the speech string, when the correct answer is only given by the one-step decision, the two-step decision gives an ambiguous answer where two decisions are in competition (zone or precise segmentation). One of them is always correct.

The rules were tested on event descriptions that are human-made. The use of these rules in a KBPR system, see [FAURE,88], where artificial instead of human perception is involved, will constitute the next step of this study. Further work will also be carried out on more complex data segmentation. Actually, the learning set has been purposely constructed to have a predominantly <Vowel-Consonant>, <Consonant-Vowel> structure, in order to test the methodology on relatively simple cases. Once the procedure for acquiring new rules has been defined and tested, it will be quite easy to apply it to more complex cases and to a richer set of training examples.

The semi-automatic methodology we have proposed is an efficient tool to obtain valuable knowledge bases and to create favorable conditions for knowledge refinement. This approach is particularly adequate when the expert task is image interpretation. Many aspects like recording the complete discourse, accepting natural language verbal data and proposing correction tasks to the expert avoid overloading her with heavy and time-consuming constraints and preserve as well as can be expected her natural framework.

REFERENCES

[BELKIN, 87] BELKIN N.J., BROOKS H.M., DANIELS P.J., "Knowledge Elicitation using Discourse Analysis", Int. J. Man-Machine Studies, 27, 127-144, (1987)

[BERGADANO, 88] BERGADANO F., GIORDANA A., SAITTA L. , "Automatic Knowledge Acquisition in Noisy Environments", IEEE Trans. on Pattern Analysis and Machine Intelligence, PAMI-10, 555-575 (1988).

[BOOSE, 87] BOOSE J. H., BRADSHAW J. M., " Expertise Transfer and Complex Problems: Using AQUINAS as a Knowledge-acquisition Workbench for Knowledge-based Systems", Int. J. Man-Machine Studies, N. 26, pp. 3-28, 1987.

[ERICSSON, 87] ERICSSON K. A., SIMON H. A., "Verbal Reports as Data", Psychological Review, Vol. 87, N.3, pp. 215-251, 1987.

[FAURE, 88] FAURE C., "Interprétation de spectrogrammes de parole", RI ENST - 88DOO1, 1988.

[FREDIANI, 87] FREDIANI S., SAITTA L., "Knowledge Base Organization in Expert Systems", in Lecture Notes in Computer Science, Vol. 286, B. Bouchon, Yager R. R. (eds), Uncertainty in Knowledge-Based Systems, pp. 217-224, 1987.

[KLINGER, 87] KLINGER G., BOYD C., GENETET S., McDERMOTT J., "A KNACK for Knowledge Acquisition", Proc. AAAI, Seattle, pp. 488-493, 1987.

[SAITTA, 88] SAITTA L., "Spectrogram Segmentation: Acquisition of the Expert's Knowledge", RI ENST - 88D010, 1988.

[WIELINGA, 87] WIELINGA B. J., BREUKER J. A., "Models of Expertise", Advances in Artificial Intelligence - II, B. du Bouley, D. Hogg, L.Steels (eds), Elsevier Science Publishers B.V. (Norh Holland), pp. 497-509, 1987.

METHODS FOR KNOWLEDGE ACQUISITION AND REFINEMENT IN SECOND GENERATION EXPERT SYSTEMS°

Nada Lavrac
Computer Science Department, Jozef Stefan Institute
Jamova 39, 61000 Ljubljana, Yugoslavia

Igor Mozetic°°
Department of Computer Science and Center for Artificial Intelligence
George Mason University, 4400 University Drive
Fairfax, VA 22030, USA

Abstract

First generation expert systems rely on the use of surface knowledge, such as associational or heuristic. Second generation technology is characterized by two additional features: *deep knowledge* and *machine learning*. Three second generation methods for knowledge acquisition are reviewed: learning rules from examples, model-based rule learning, and semi-automatic model acquisition. The man-machine process of acquiring and refining knowledge extends the role of expert systems to *expert support systems*, since both man and machine learn through repeated knowledge refinement cycles. Explanation of solutions and of the knowledge base itself is crucial for this man-machine learning process. An extended expert system shell schema is presented that includes a knowledge acquisition and a knowledge explanation module.

1 Introduction

First generation expert systems rely on *surface*, or shallow, operational knowledge acquired through a process of direct articulation. On the other hand, the trend of the second generation expert systems is to include *deep* knowledge as well, thus capturing the underlying causal structure of the problem domain (Steels 1985), and to at least partially automatize the knowledge acquisition process by using machine learning techniques (Michie *et al.* 1984). We review three methods for knowledge acquisition in expert systems: learning rules from examples, model-based rule learning, and qualitative model acquisition. All the methods were used in the development of a medical expert system KARDIO for the ECG diagnosis of cardiac arrhythmias (Bratko, Mozetic & Lavrac 1988, 1989). The reviewed methods can be considered as second generation knowledge acquisition methods, and could be incorporated into a knowledge acquisition module of a second generation expert system shell.

The process of acquiring knowledge typically requires several 'refinement cycles' in which the machine's performance is compared with the human expert's performance. Also, the

°The paper was published in the special issue of the ACM SIGART Newsletter on knowledge acquisition, 1989.
°°Current affiliation: Vakgroep C.I.S., Faculteit Informatica, Universiteit Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands.

machine representation of knowledge is compared with the original expert's knowledge. In the refinement cycles, both man and machine will learn, provided that the machine's representation of knowledge is compact and comprehensible. This man-machine learning process that consists of acquiring and refining knowledge, extends the role of expert systems to *expert support systems* (Luconi, Malone & Scott Morton 1986). Explanation of the solutions and of the knowledge base itself is crucial for this man-machine learning process. Bohanec, Rajkovic & Lavrac (1988) introduce an extended schema of an expert system shell that includes a knowledge acquisition and a knowledge explanation module. The task of the knowledge acquisition module is to support user-friendly encoding of knowledge as required by one of the reviewed methods. The task of the knowledge explanation module is not just to display knowledge in an understandable form, but also to represent knowledge from different viewpoints and at different levels of detail.

2 Second generation methods for knowledge acquisition

Knowledge acquisition is typically a demanding mental process, where a knowledge engineer collaborates with domain experts. In this process the knowledge engineer's objective is to convert human know-how into 'say-how' through a process of articulation. *Direct encoding* of rules, semantic nets, frames, etc., encounters 'the bottleneck problem of applied artificial intelligence' (Feigenbaum 1977) and is named the 'old style knowledge engineer's route map' by Michie (1986). In Figure 1, which shows the flow of knowledge in different knowledge acquisition paradigms, this 'route map' corresponds to the knowledge flow through box A. Recently, there have been a number of interview techniques (Welbank 1983, Hart 1986) and software tools developed, e.g., AQUINAS (Boose & Bradshaw 1987), KADS (Breuker &

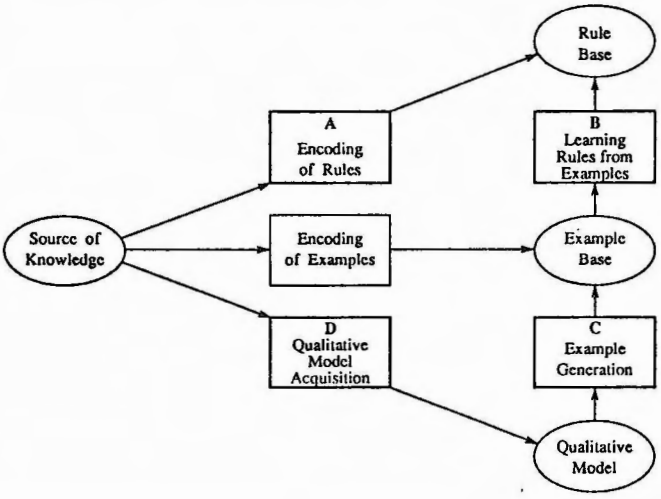


Figure 1. Knowledge flow in different methods for knowledge acquisition.

Wielinga 1987), MORE (Kahn, Nowlan & McDermott 1985). For the most part, they use a conceptual model to interact with the user thus hiding the complexity and unfamiliarity of the model (rules, nets, ets.) upon which the knowledge base is actually constructed.

2.1 Learning rules from examples

Second generation knowledge acquisition methods are characterized by two features: use of deep knowledge, and application of machine learning techniques. The method of *learning rules from examples* is recognized by Michie (1986) as a 'new style knowledge engineer's route map' where rules are elicited from the expert to the machine memory via the language of examples rather than via the language of explicit articulation. Effective algorithms for inductive inference are required. There are a number of inductive learning programs, such as the programs of the TDIDT family (top-down induction of decision trees, e.g., Quinlan 1986), or the AQ family (Michalski *et al.* 1986) that accept tutorial examples and induce knowledge in the form of decision trees or rules, respectively. In Figure 1 this process is represented by box B where the source of knowledge is either an expert, formulating a series of thoroughly chosen examples, or preferably an existing database of examples interpreted and 'cleaned' with the help of an expert. Results of applying machine learning systems, i.e., ASSISTANT (Bratko & Kononenko 1987) and AQ15 (Michalski *et al.* 1986), on several real-life medical domains show that the systems' predictive accuracy is at the level of the best domain experts (Table 1).

Medical domain	ASSISTANT	AQ15	Medical specialists	
Lymphography	77%	82%	85%	estimate
Breast cancer	72%	68%	64%	5
Primary tumor	46%	41%	42%	4

Table 1. Diagnostic accuracy of the learning programs ASSISTANT and AQ15, averaged over 4 experiments, as compared to medical specialists. The last column denotes the number of specialists tested.

2.2 Model-based rule learning

Second generation knowledge acquisition methods are also concerned with the elicitation of *deep* knowledge that captures the underlying causal structure of the problem domain. Such knowledge can be represented in the form of a model that states the 'first principles' or basic 'rules of the game' from which operational decisions can be derived. The prevailing type of knowledge in such a model is *qualitative* (de Kleer & Brown 1984; Forbus 1984; Kuipers 1986; Bratko, Mozetic & Lavrac 1989). This has several advantages over conventional numerical modeling: the qualitative view is often closer to reasoning about the physical or physiological processes being modeled; to execute the model we do not have to know the exact numerical values of the parameters in the model; a qualitative simulation may be computationally less complex than numerical simulation. A qualitative simulation can be used for the explanation of the mechanisms of a system which is being modeled more naturally than in numerical modeling.

In principle, a qualitative model can be used for problem solving directly. It can answer prediction, diagnostic, and control type of questions. The *prediction* task is to find the observable results of applying some input to the system, given a functional state of the system. The *diagnostic* task is: given the inputs to the system and the observable manifestations, find the system's functional state (normal or faulty, which components are failed). The *control* task is to determine the input control to the system, assuming its state, in order to achieve a desired output.

However, a model is primarily designed for simulation and prediction. Using it to solve diagnostic and control tasks might be computationally expensive. On the other hand, by exhaustive qualitative simulation, the model can be used to automatically generate examples of any possible behavior. From such an exhaustive set of examples operational decision rules can be generated by inductive learning methods. So, what we call *model-based rule learning* consists of two steps: example generation using a qualitative model (box C in Figure 1), and learning rules from the automatically generated examples (box B in Figure 1). This knowledge acquisition paradigm has been used in KARDIO to generate compressed diagnostic and prediction rules as shown in Figure 2. We have developed a deep qualitative model of the electrical activity of the heart, and have used it for the automatic synthesis (through simulation) of the surface knowledge about the ECG interpretation. This has the form of pairs (Arrhythmia, ECG description) relating one of the 2,419 possible combined arrhythmias to the corresponding ECG patterns (there are altogether 140,966 ECG patterns). The surface representation facilitates fast ECG diagnosis, but is rather complex in terms of memory space (over 5 MB, stored as text file). This motivated the compression of the surface knowledge by means of an inductive learning program

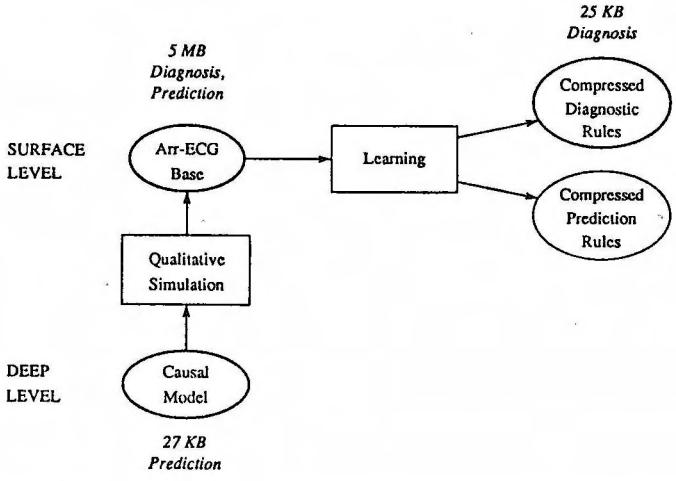


Figure 2. Deep and surface levels of cardiological knowledge, and transformations between representations.

of the AQ family into a compact and diagnostically efficient representation (Mozetic 1986). In Figure 2 the representations are arranged as to emphasize the distinction between the deep and surface levels of knowledge.

In KARDIO, we demonstrated how the qualitative modeling approach and the machine learning technology can be used to construct knowledge bases with complexity far beyond the capability of traditional dialogue-based techniques for knowledge acquisition. The KARDIO knowledge acquisition paradigm, described in detail in (Bratko, Mozetic & Lavrac 1989) may become a standard technique in the development of practical expert systems. It has already been used in the development of a satellite power supply fault diagnosis system by Pearce (1988).

2.3 Qualitative model acquisition

The model design process can be at least partially automated by means of machine learning. A Qualitative Model Acquisition System (QuMAS), described in (Mozetic 1987a, b; Bratko, Mozetic & Lavrac 1989), supports the construction of a deep model, and the representation of a model at different levels of detail. In QuMAS, partial knowledge about the model and examples of its behavior are provided by the user, and the complete model is automatically constructed and incrementally refined until the desired behavior is achieved. This knowledge acquisition paradigm is represented by box D in Figure 1.

In this approach, we restrict ourselves to functional qualitative models, where a model is defined by its structure (a set of components and their connections) and functions of the individual components. QuMAS consists of three subsystems: a *learner* that hypothesizes functions of components from examples of their behavior, an *interpreter* that can use the hypothesized model to derive its behavior, and a *debugger* that locates faulty functions of components and proposes how to correct them. It is assumed that partial knowledge about the model is given - its structure. Further, examples of the behavior of the model and its constituent components are provided from which the learning part of the system hypothesizes functions of the components. The interpreter of the model is then able to derive its behavior. The user may test the model and compare it with the intended behavior. When a difference between the derived and the intended behavior of the model occurs, a debugger is invoked. The debugger locates faulty hypotheses defining functions of components, proposes examples of behavior that guarantee the intended behavior of the model, and invokes the learner that incrementally refines the hypotheses. The cycle of deriving the behavior of the model, debugging the model, and incremental learning is repeated until the intended behavior of the model is achieved, i.e., the user believes that the model is correct and complete with respect to the actual system being modeled. An overview of QuMAS is in Figure 3a.

QuMAS embodies two types of learning. Initial data-driven learning generalizes examples of components' behavior into rules on the basis of similarities and differences and does not require any user interaction. The second type of learning is model-driven where the debugger actively constructs examples of components' behavior which satisfy the intended model behavior, and then queries the user for confirmation. QuMAS therefore offers a tradeoff between the initial amount of knowledge provided by the user, and the time one is willing to spend on debugging the model. QuMAS is used interactively by the model designer, and takes

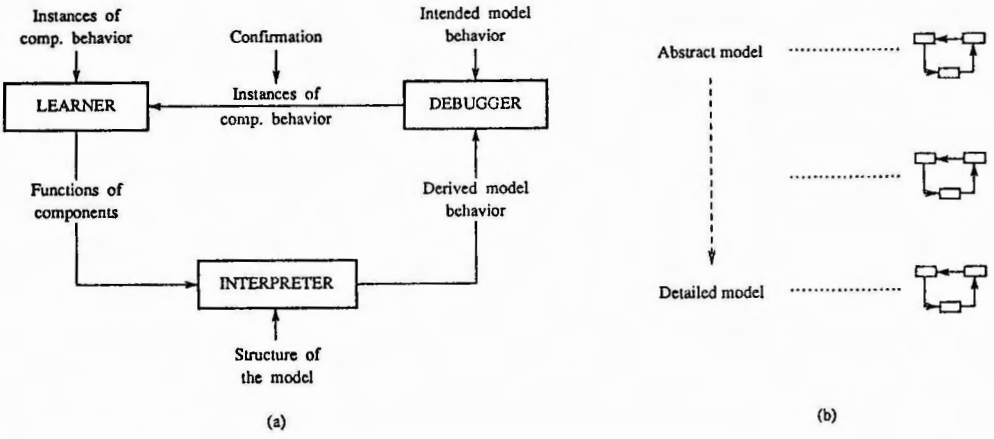


Figure 3. An overview of the qualitative model acquisition system (a), and the top-down model construction method (b).

advantage of the hierarchical model representation to speed up the automatic learning of the model (Figure 3b). The hierarchy also has a role in generating good and concise explanations on points selected by the user. A substantial submodel of the KARDIO heart model was reconstructed semi-automatically using QuMAS (Mozetic 1987a, b; Bratko, Mozetic & Lavrac 1989).

3 The role of refinement cycles in knowledge acquisition

Development of an expert system is essentially an iterative process that typically needs several *refinement cycles*. In each cycle an expert and a knowledge engineer refine the knowledge base by comparing the human performance with the machine performance, and the original human knowledge with the generated machine representation. The refinement cycles need to be repeated until the intended performance of the system is achieved.

Expert system shells have to integrate a variety of tools that allow for acquisition, explanation and utilization of complex domain knowledge. The classical expert system schema (Figure 4a) cannot cover all the required functions of the system. In this schema, an expert system consists of a domain dependent knowledge base, and of a domain independent expert system shell which incorporates an inference engine and a user interface. In Figure 4b we propose a new schema of an expert system shell. Here the structure of the shell is extended to incorporate: the *knowledge acquisition module* which provides tools for acquiring and editing the knowledge base, the *knowledge explanation module* allowing for different representations of the knowledge base, and the *knowledge utilization module* which applies the knowledge base to find solutions to a problem. It also has to provide explanations of particular solutions, and enable the knowledge base validation.

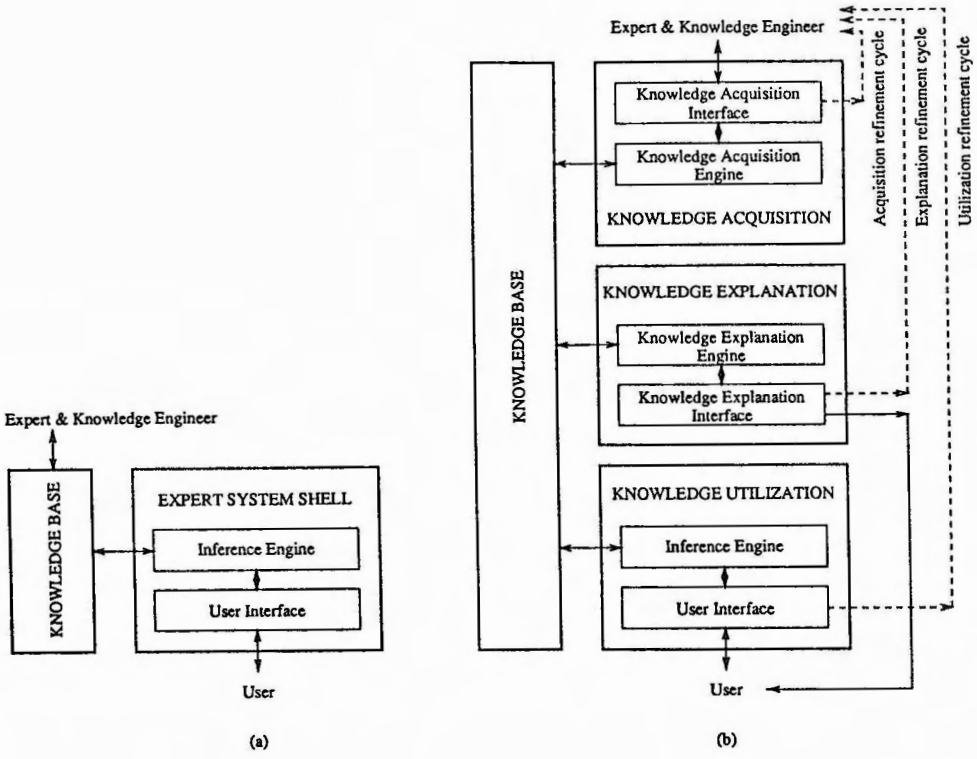


Figure 4. A classical (a) and an extended schema (b) of an expert system shell.

The new schema covers all functional requirements of expert systems as recognized by Gaines (1987), namely, apply, explain, acquire, display, edit, and validate the knowledge base. According to our view, an expert system shell is not aimed just to support problem solving, but should also actively support knowledge acquisition and refinement of the acquired knowledge. Therefore two new modules are introduced: the knowledge acquisition and the knowledge explanation module.

The knowledge acquisition module is to support one or more methods for knowledge acquisition described in the previous section. It has to incorporate a machine learning system and/or a system that supports direct or semi-automatic construction of qualitative models. The knowledge explanation module is aimed at generating different explanations - from displaying the knowledge base in a compact and comprehensible form, to representing the knowledge from different viewpoints and at different levels of detail. This may be achieved by varying the syntax and the semantic of the representation and by varying the level of detail (Bohanec, Rajkovic & Lavrac 1988). By a 'different syntax' we mean the representation of the same knowledge in a different language, e.g., a graphical or tabular representation. By a 'different semantic' we

understand either the reorganization of knowledge (e.g., grouping rules, or expressing one set of rules with another), or the representation of additional information derived from the original knowledge base (e.g., different statistics, Bayesian probabilities, informativity of attributes, etc.). By a 'different level of detail' we mean knowledge representation by using attributes at a chosen level of the taxonomic hierarchy, or by eliminating too specific knowledge, e.g., presenting only the most important rules. Some of these features are provided in the machine learning systems ASSISTANT and AQ15, mentioned in the previous section.

According to our expert system shell schema, we distinguish between the *acquisition*, *utilization* and *explanation refinement cycle* (represented by dashed lines in Figure 4b). These refinement cycles have to be supported by appropriate development tools.

Depending on the type of the system's knowledge base, the *acquisition refinement cycle* in Figure 4b consists of a feedback loop from any type of acquired knowledge back to the source of knowledge (see Figure 1). In the case when a knowledge base is induced from a set of tutorial examples this corresponds to Michie's 'first refinement cycle' of his 'new style knowledge engineer's route map' (Michie 1986). His 'second refinement cycle' corresponds to what we call here the *utilization refinement cycle* in which the system's performance is compared with the human expert's performance.

The knowledge explanation module is aimed to provide different representations of the knowledge base. This motivates the expert and the knowledge engineer to further check and elaborate the knowledge base as new ideas are triggered that may lead to finding inadequate or missing knowledge. In the process of knowledge acquisition we call this the *explanation refinement cycle*. In this cycle, the generated machine representation of knowledge is compared with the original expert's knowledge. Differences between the two can result from either an error in the machine representation, or a slip in the original human codification of knowledge. In the former case, the error is a consequence of incorrectly encoded rules, errors in learning examples, or an error in the deep causal model, depending on the type of the system's knowledge base. In the latter case, the refinement cycles may expose blemishes in the existing, original expert formulations, and may thus help to improve them.

In the man-machine dialogue through several refinement cycles both man and machine learn (Chambers & Michie 1969). This enables considering the process of knowledge elicitation as a man-machine learning process consisting of the stepwise acquisition and refinement of knowledge. The level of human and machine knowledge grows. From this point of view, the role of an expert system is not only in solving problems and explaining the solutions, but also in building (new) human-type knowledge through refinement cycles. By acquiring this refined knowledge the machine knowledge is improved and allows for better performance. This man-machine learning process that consists of acquiring and refining knowledge, extends the role of expert systems to *expert support systems* (Luconi, Malone & Scott Morton 1986). In contrast to expert systems, where main goal is to simulate experts' performance in problem solving, expert support systems stimulate human mental processes in the process of knowledge acquisition, and therefore supporting human learning as well.

4 Conclusion

The development of an expert system is typically an iterative process. It consists of refinement cycles that are repeated until the intended performance of the system is achieved. In the utilization refinement cycle, a machine's performance is compared with the human expert's performance. In the acquisition and in the explanation refinement cycle, the generated machine representation of knowledge is compared with the original expert's knowledge. The three refinement cycles are named according to the three modules proposed in the extended expert system shell schema. We emphasize the importance of the knowledge acquisition, and the knowledge explanation cycle. They allow seeing the development of an expert system as a process in which both man and machine learn, and in which (new) human-type knowledge is generated. The reviewed methods for knowledge acquisition and refinement, all actually used in KARDIO to model a substantial real-life problem, emphasize the role of machine learning techniques in the development of second generation expert systems.

Acknowledgements

All the described methods for knowledge acquisition were used (and two of them developed) when building KARDIO, a medical expert system for the ECG diagnosis of cardiac arrhythmias, in collaboration with Ivan Bratko. The extended expert system shell schema was proposed in collaboration with Marko Bohanec and Vladislav Rajkovic. We are grateful to Donald Michie for his valuable comments on an earlier draft of the paper.

This research was supported in part by the Slovenian Research Council, the COST 13 project Machine Learning and Pattern Recognition, the Defense Advanced Research Project Agency under grant, administered by the Office of Naval Research, No. N00014-87-K-0874, the Office of Naval Research under grant No. N00014-88-K-0226, and the Office of Naval Research under grant No. N00014-88-K-0397.

References

Bohanec, M., Rajkovic, V., & Lavrac, N. (1988) Knowledge explanation in expert systems: A decision support system and machine learning view. *Proc. IASTED Intl. Conference on Expert Systems*, Geneve, Switzerland.

Boose, J., & Bradshaw, J.M (1987) AQUINAS: A knowledge acquisition workbench for building knowledge-based systems. *Proc. First European Workshop on Knowledge Acquisition for Knowledge-based Systems*, Reading University.

Bratko, I., & Kononenko, I. (1987) Learning diagnostic rules from incomplete and noisy data. In: Phelps, B. (ed.) *AI Methods in Statistics*, London: Gower Technical Press.

Bratko, I., Mozetic, I., & Lavrac, N. (1988) Automatic synthesis and compression of cardiological knowledge. In: Hayes, J., Michie, D., & Richards, J. (eds.) *Machine Intelligence 11*, Oxford: Oxford University Press. Also in: Michie, D., & Bratko, I. (eds.) *Expert systems: Automating knowledge acquisition*, Addison-Wesley, 1986.

Bratko, I., Mozetic, I., & Lavrac, N. (1989) *KARDIO: A study in deep and qualitative knowledge for expert systems*. Boston: The MIT Press (in press).

Breuker, J., & Wielinga, B. (1987) Knowledge acquisition as modeling expertise: The KADS methodology. *Proc. First European Workshop on Knowledge Acquisition for Knowledge-based Systems*, Reading University.

Chambers, R.A., & Michie, D. (1969) Man-machine co-operation on a learning task. In: Parslow, R.D., Prowse, R.W. & Elliot-Green, R. (eds.) *Computer Graphics: Techniques and Applications*, London: Plenum Press.

de Kleer, J., & Brown, J.S. (1984) A qualitative physics based on confluences. *Artificial Intelligence 24 (1-3)*, pp. 7-83.

Feigenbaum, E.A. (1977) The art of artificial intelligence: Themes and case studies of knowledge engineering. Pub. No. STAN-SC-77-621, Stanford, CA: Stanford University, Department of Computer Science.

Forbus, K.D. (1984) Qualitative process theory. *Artificial Intelligence 24 (1-3)*, pp. 85-168.

Gaines, B.R. (1987) Knowledge acquisition for expert systems. *Proc. First European Workshop on Knowledge Acquisition for Knowledge-based Systems*, Reading University.

Hart, A. (1986) *Knowledge acquisition for expert systems*. London: Kogan Page.

Kahn, G., Nowlan, S., & McDermott, J. (1985) MORE: An intelligent knowledge acquisition tool. *Proc. Ninth Intl. Joint Conference on Artificial Intelligence*, Los Angeles, CA: Morgan Kaufmann.

Kuipers, B. (1986) Qualitative simulation. *Artificial Intelligence 29*, pp. 289-338.

Luconi, F.L., Malone T.W., & Scott Morton, M.S. (1986) Expert Systems: The next challenge for managers. Sloan Management Review.

Michalski, R.S., Mozetic, I., Hong, J., & Lavrac, N. (1986) The multi-purpose incremental learning system AQ15 and its testing application on three medical domains. *Proc. Natl. Conference on Artificial Intelligence, AAAI-86*, pp. 1041-1045, Philadelphia: Morgan Kaufmann.

Michie, D. (1986) Machine learning and knowledge acquisition. In: Michie, D., & Bratko, I. (eds.) *Expert systems: Automating knowledge acquisition*, Addison-Wesley. Also published as Current developments in expert systems. In: Quinlan, J.R. (ed.) *Applications of Expert Systems*, Addison-Wesley, 1987.

Michie, D., Muggleton, S., Riese, C. & Zubrick, S. (1984) RULEMASTER: A second-generation knowledge-acquisition facility. *Proc. First Conference on Artificial Intelligence Applications*, pp. 591-597, Denver.

Mozetic, I. (1986) Knowledge extraction through learning from examples. In: Mitchell, T.M., Carbonell, J.G., & Michalski, R.S. (eds.) *Machine Learning: A Guide to Current Research*, pp. 227-231, Boston: Kluwer Academic Publishers.

Mozetic, I. (1987a) Learning of qualitative models. In: Bratko, I., & Lavrac, N. (eds.) *Progress in machine learning*, pp. 201-217, Wilmslow, UK: Sigma Press.

Mozetic, I. (1987b) The role of abstractions in learning qualitative models. *Proc. Fourth Intl. Workshop on Machine Learning*, pp. 242-255, Irvine, CA: Morgan Kaufmann.

Pearce, D.A. (1988) The induction of fault diagnosis systems from qualitative models. *Proc. Natl. Conference on Artificial Intelligence, AAAI-88*, pp. 353-357, Saint Paul, MN: Morgan Kaufmann.

Quinlan, J.R. (1986) Induction of decision trees. *Machine Learning 1 (1)*, pp. 81-106.

Steels, L. (1985) Second generation expert systems. *Future Generation Expert Systems 1 (4)*, pp. 213-221.

Welbank, M. (1983) A review of knowledge acquisition techniques for expert systems. Ipswich: Martlesham Consultancy Services.

Integrating Acquisition, Representation and Interpretation

Marc Linster
Expert System Research Group
GMD
PO Box 1240
D-5205 St. Augustin 1

e-mail : linster@gmdzi.uucp

Introduced to EKAW89

We motivate the need for the match of problem-solving strategies and knowledge-acquisition tools in a task-independent knowledge-acquisition environment. We present IRA-grid, a prototypical acquisition system that acquires some of the knowledge needed for classification tasks. To that purpose IRA-grid uses repertory-grid-like procedures to acquire knowledge, transforms this knowledge into a more systematic attribute-value representation, and then allows the user to see this knowledge act, and thus understand what is really in it.

We briefly present IRMA, a future environment that will include several tools of the IRA-type, together with modeling and interpretation facilities, thus allowing to specify problem-solving procedures, which will in turn trigger knowledge acquisition tools that provide the symbol-level knowledge needed for the problem-solving procedures.

1. VIEWING KNOWLEDGE ACQUISITION AS A CONSTRUCTION PROCESS

Knowledge acquisition is classically viewed as the transfer of knowledge (Feigenbaum 1980) from experts to machine-interpretable data-structures. To realize this transfer process interview and protocol techniques are being used (Welbank 1983). It is our opinion however that when interviewing or observing experts we do not acquire expertise, all we get is the manifestation of the expert's knowledge in his answers and in his behavioral patterns.

It is our point that:

- knowledge cannot be acquired from experts in interviews or through protocol techniques. All we can acquire are bits of knowledge, represented through single, observable actions, which, when combined in the right way, represent the expert's knowledgeable behavior;

- the bits of knowledge, which represent building blocks for intelligent behavior cannot be viewed without at the same time considering the way they will be used lateron in a problem solving process, i.e. manifestations of knowledge are not meaningful without their context.

If we define the building of expert systems as the problem of creating an operational formalism that behaves like an expert and shows expert-like behavioral patterns in a limited domain and under well-specified circumstances, then knowledge acquisition must be redefined as a construction process. We acquire knowledge bits, we describe their role in a problem-solving process, and to see if we acquired knowledge, i.e. if our operationalized bits of knowledge really behave knowledgeable, we trigger an interpreter that combines them and shows their behavior.

We call this view the construction paradigm of knowledge acquisition:

> *Knowledge acquisition is concerned with the collection of manifestations of human knowledge, description of the roles of the building-blocks of these manifestations and the construction of problem-solving behavior from these building-blocks.*

A tool that supports this view must integrate acquisition, knowledge-modeling and interpretation facilities.

2. SEVERAL KINDS OF KNOWLEDGE

To develop knowledge-based systems according to the construction-paradigm we must differentiate between two kinds of knowledge at least: the knowledge contained in the building-blocks of our system and the knowledge concerned with the description and the use of these building-blocks. Newell (Newell 1982) called them symbol-level and knowledge-level, and proposed to use the knowledge-level as a description of the symbol-level elements (Newell 1982, p. 115).In KADS (Knowledge Acquisition and Design Structuring) (Breuker et al 1987) the knowledge-level is split into several levels to allow for a systematic description of the roles of knowledge-bits, the interaction of the bits and the interaction of the roles (Fig. 1).

Meta-classes describe the roles of the domain concepts, e.g. if a concept is an observable or a hypotheses, a symptom or a solution. Knowledge sources are primitive inferences that are applied to meta-classes, e.g. abstraction and heuristic match are knowledge sources that appear in diagnostic problem-solving. Tasks describe how inference-level elements are combined to form problem-solving behavior. The strategic level is concerned with knowing about problem-solving techniques, their usability and their knowledge-needs.

Comparing KADS and Newell's knowledge-levels we see that the KADS domain-level is quite like the symbol-level of a traditional expert system, and that the inference and task-layers are the

corresponding knowledge-levels. But in addition to this, task- and inference-layer are symbol-level to the strategic-level, which is in turn their knowledge-level. KADS allows thus for reflective systems.

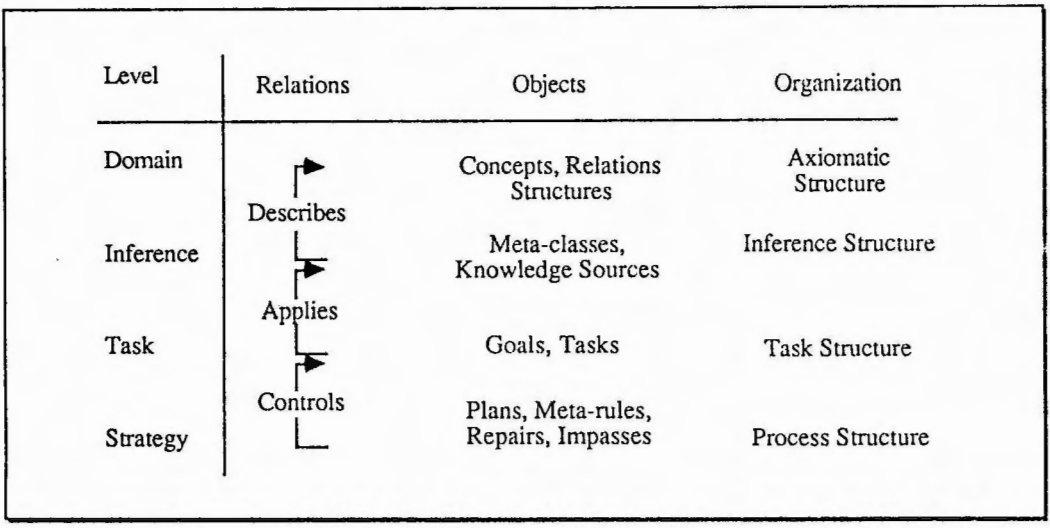


Fig. 1: Knowledge-modeling levels in KADS (Breuker et al 1987, p. 11)

If we accept Newell's view on knowledge we see that the results of most knowledge acquisition techniques are limited to symbol-level knowledge. In repertory-grids (Gaines & Shaw 1981) we get information on classes and entities described by these classes. Protocol-analysis (Ericsson & Simon 1984, Diederich 1987) gives us information about steps and situations occurring in one or more problem-solving processes, systemic grammars (Johnson & Tomlinson 1988) record sequences of choices. None of these techniques give us knowledge-level information, i.e. information about what really happens. Knowledge-level information tells us: this is diagnosis, that is classification, these steps correspond to hypotheses generation a.s.o. Thus the knowledge-level describes the actions, goals and purposes of a system.

The only system, known to the author of this article, that allows explicit formulation of knowledge-level information, and that uses this information to acquire the necessary symbol-level-knowledge needed to fulfill the task described at the knowledge-level, is PROTEGE (Musen 1988a; Musen 1988b). But the acquisition tools that are used aren't knowledge elicitation tools, they are knowledge editors that use forms that the expert fills in.

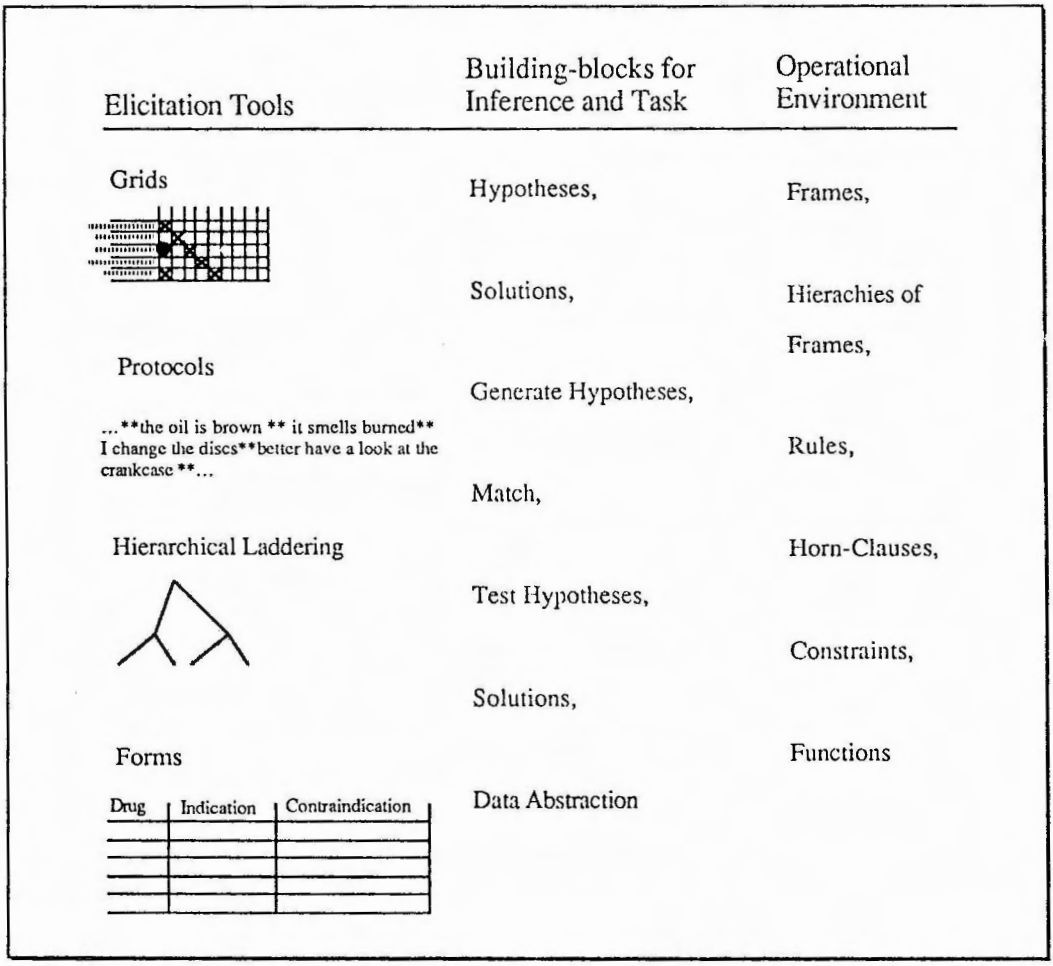


Fig. 2 : The problem of matching elicitation tools to tasks and those to operational representation mechanisms. Matching these three means defining the operationaliztion of a knowledge-source, i.e. match, in terms of the operational environment, i.e. Prolog, and the definition of an acquisition tool, i.e. protocol analysis, to acquire the knowledge needed for the knowledge-source.

3. THE NEED TO INTEGRATE THE ACQUISITION OF DOMAIN- AND INFERENCE-LEVEL KNOWLEDGE

All automated knowledge acquisition systems known to the author are specialized tools. They acquire knowledge for one problem-solving method. Tools like OPAL constrain themselves to one problem-solving method in one domain.

No efforts have gone towards the acquisition of task- or inference-knowledge. Some concern has gone to the representation of such knowledge though, e.g. generic tasks (Bylander & Chandrasekaran 1986) or KADS.

It seems promising to use explicit representations of problem-solving methods to guide symbol-level acquisition tools like grids, preference analysis (Bradshaw & Boose 1988) etc. That way we could build truly domain- and task-independent knowledge-acquisition tools.

4. PERFORMANCE AND ACQUISITION : UNDERSTANDING WHAT'S REALLY IN IT

As stated above we see knowledge acquisition as a process of creation of operational knowledge. We distinguished three kinds of knowledge, two of which seem relevant for today's and tomorrow's knowledge-based systems : domain- and task/inference-knowledge. Strategic knowledge will be important for reflective systems. A set of knowledge acquisition tools exists for the acquisition of domain- or symbol-level knowledge. Representations for higher levels are under development.

It is our point of view that we must merge acquisition, representation of problem-solving methods and interpretation of the knowledge in one environment to create general-purpose knowledge-acquisition, -representation and -interpretation tools.

5. MATCHING PROBLEM-SOLVING METHODS TO KNOWLEDGE ACQUISITION METHODS

If we want to develop knowledge acquisition environments that are task-independent, i.e. that are not restricted to classification, diagnosis or skeletal-plan refinement, then we need environments that allow us to specify tasks, and that can then acquire the knowledge needed to fulfill these tasks. (McDermott 1989) calls this the need to define the *computational requirements of tasks*. In the terminology of this paper we express this as the need of a mapping between knowledge-level descriptions and the knowledge acquisition tools that deliver the symbol-level knowledge.

The experimental tool IRA-grid that I will present in the next section is a tool that acquires some of the knowledge that a knowledge-source concerned with classification needs. It uses repertory-grid and a simple inference scheme to acquire a rich description of the potential solutions, and many different solutions that have comparable descriptions. The knowledge needed for the procedure of classification, i.e. which attribute is inquired about first and which comes later, is the domain of other knowledge acquisition techniques like protocol analysis or preference analysis.

6. IRA-GRID

IRA-grid (Interpretation, Representation and Acquisition using repertory-grids) is a knowledge acquisition tool that acquires some of the knowledge needed for classification tasks.

IRA-grid is a first step towards acquisition of operational knowledge guided by the needs of a problem-solving method.

To elicit knowledge IRA-grid uses repertory-grids. The information from the grids is then transformed into a more systematic attribute-value representation, implication-rules are induced, constraints are defined on the attribute-values, and the rules, the constraints and the concepts can be tested in the general-purpose representation and inference environment BABYLON. For the interpretation of the knowledge in BABYLON a very simple inference scheme is used, as the knowledge needed for the classification procedure itself, i.e. which attribute to use first a.s.o., is subject of other acquisition procedures.

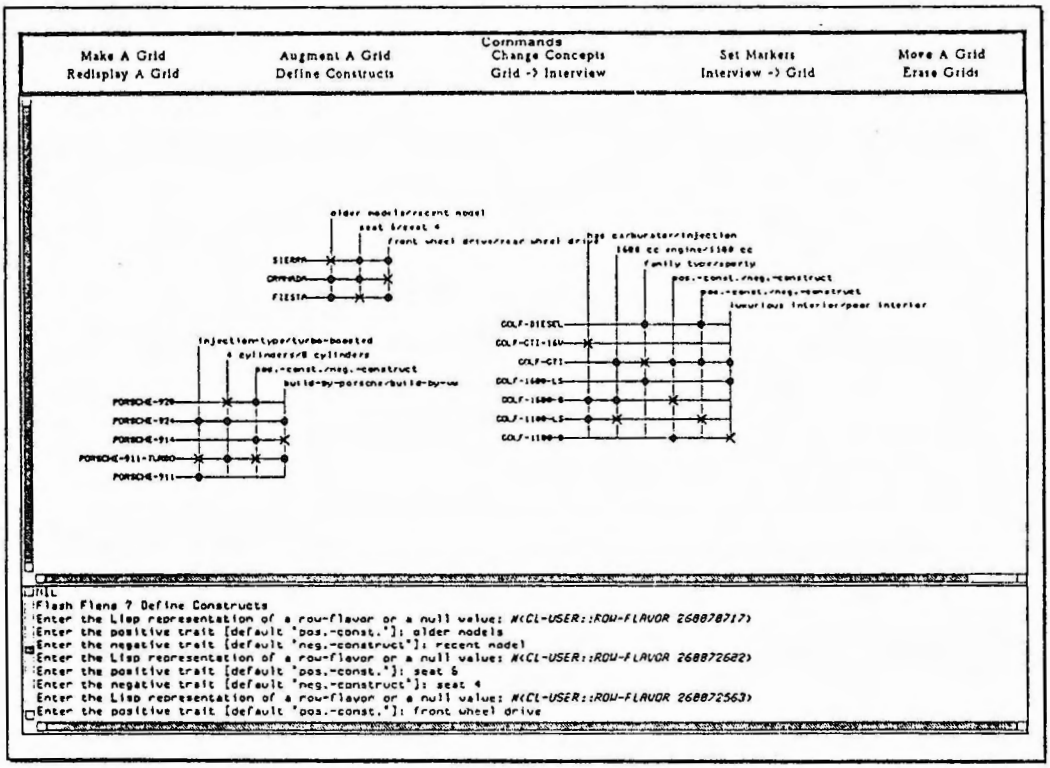


Fig. 3 : Three grids on cars.

O stands for "positive descriptor is applicable", X means "negative descriptor is applicable"

The first question (i.e. the first column) in the grid on the lower left can be read as :

What do Porsche-924 and Porsche-911 have in common that differentiates them from Porsche-911-Turbo? The answer in the grid is: The first two are injection-type, the third one is turbo-boosted.

6.1. Acquisition

The elicitation component (fig. 3) acquires concepts and descriptors for these concepts. Descriptors can be arbitrary strings. The representation that was chosen for the grids is close to the representation that George Kelly (Kelly 1955, p. 302) chose for his tests.

We used this matrix-like graphic so that the user can compare his answers, and keep them consistent throughout the grid, and because we did not want to impose any ordering among the questions. Experience has shown that the first answers in a grid are the hardest, and that the next questions are answered in a comparable fashion, i.e. using distinctions coming from the same context.

Several grids can be represented and edited at the same time.

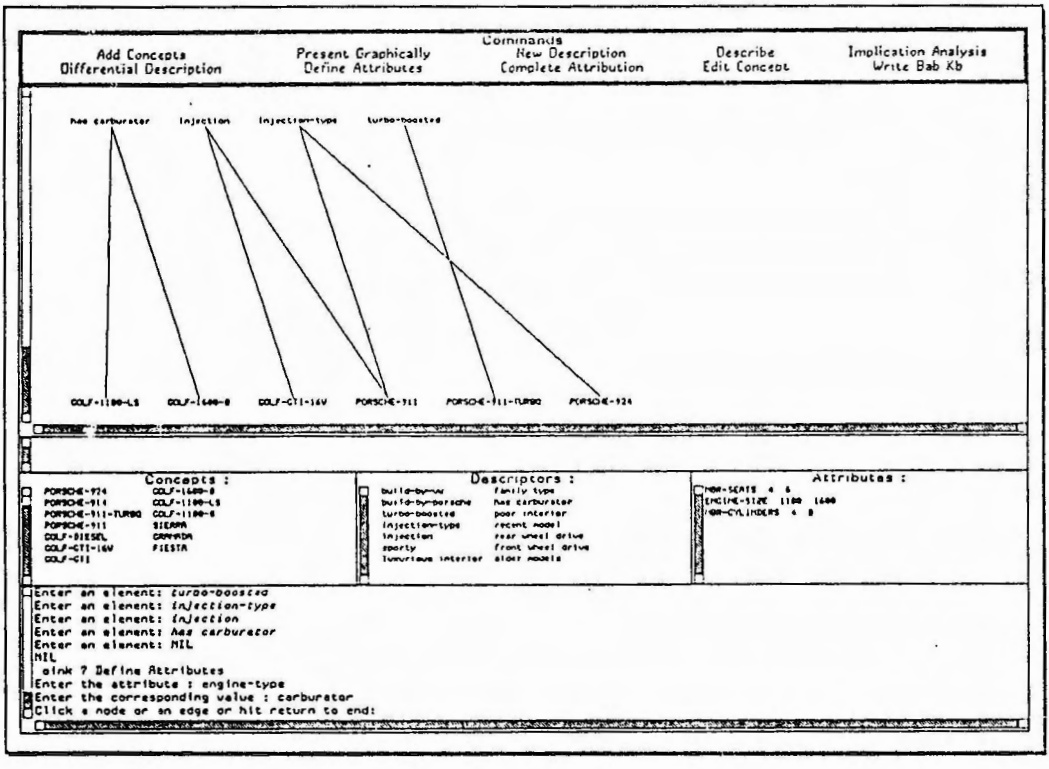


Fig 4. : The descriptors of the concepts are changed into attribute-value representations

6.2. Representation

Descriptors are a suitable way to elicit knowledge, but attributes and values of these attributes are more systematic and allow for more precise descriptions. In the representation phase we allow the user of IRA-grid to define attributes and values of these attributes for certain concepts (fig. 4). These attributes and their values then replace the descriptors. That allows the user to give his formulations a

second thought. We found out that users of the KRITON-system (Linster 1988), who had to formulate their knowledge right away as attribute-values, frequently made inconsistent use of their descriptions of concepts.

Only after describing many elements of a domain, users found that they used descriptions according to the local context. It is evident that in every elicitation session this local context changes. What we need for knowledge-based systems though is one consistent use of every descriptor or concept throughout the system. Thus, both the possibility that the experts have to reconsider statements after the elicitation phase, and the ability to freely express themselves during the elicitation phase promises to acquire more systematic knowledge.

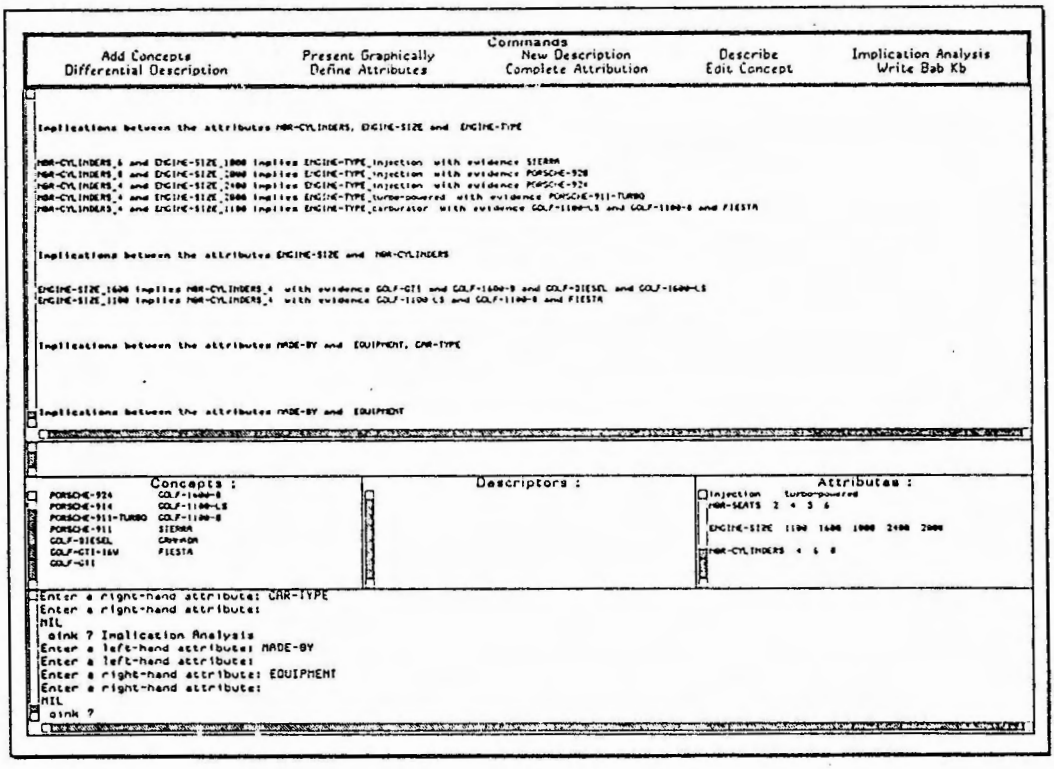


Fig 5. : Rules have been induced from the attribute-value descriptions of the concepts.

Some of the features the system offers are :

- change some or all occurrences of a descriptor into an attribute-value representation,
- view and edit the acquired knowledge,
- induce inference rules (fig 5.). This feature uses all the concepts and their attribute-value descriptions and induces all the implications that are valid. This is a complex process that finds all

wanted and unwanted regularities among the descriptions. The rules are documented by the concepts that support them. A rule can thus be removed from the valid set of rules by changing one of the supporting concepts, or by adding to the knowledge-base a new concept who's description is inconsistent with the contents of the rule.

The rule left- and right-hand side can be specified. Experience has shown that left-hand sides with more than two attributes are hard to understand for the user.

- write a BABYLON knowledge-base.

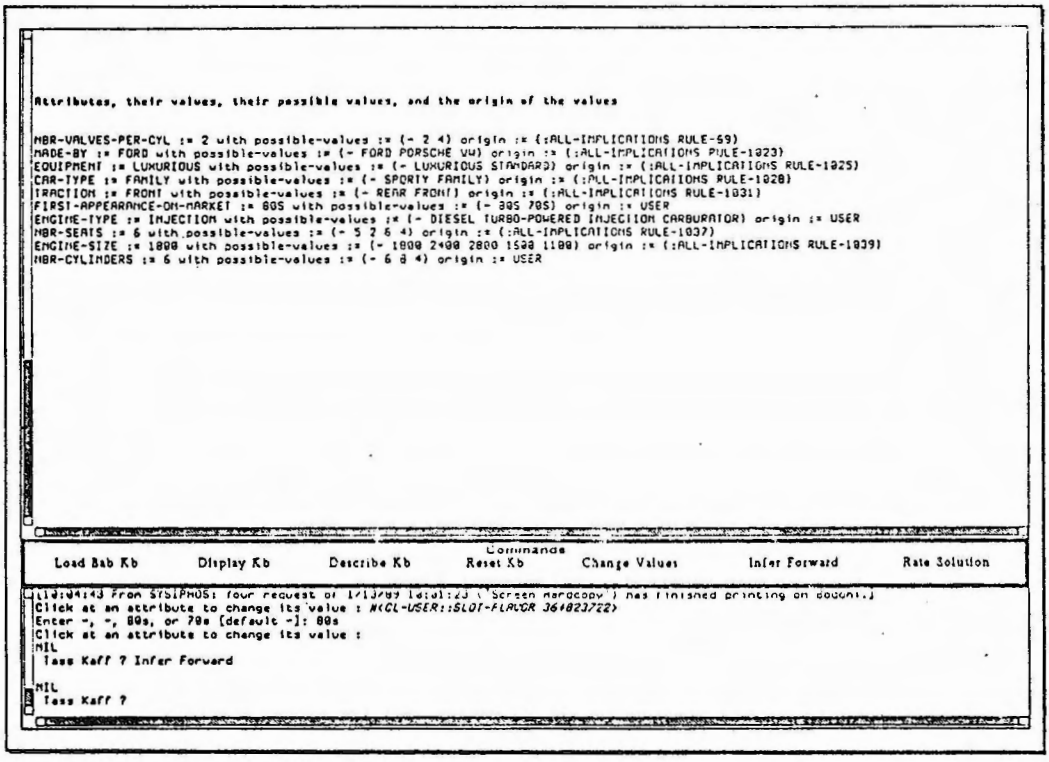


Fig. 6 : Testing the operational knowledge in a classification environment

6.3. Interpretation

BABYLON (Di Primio & Wittur 1987) is a hybrid tool based on frames, forward- and backward-rules, a subset of prolog and constraints (Güsgen, Junker & Voss 1987). The test of knowledge acquired from the repertory-grids is currently limited to a simple classification-like scheme. All the concepts and their descriptions are represented as instances of a frame, as slots and as values. The rules induced in the representation phase are used as forward inference rules defined on the instances of the frame. Constraints are generated that define consistent attribute-value sets, i.e. which value of engine-size is consistent with which value of nbr-cylinders. An extra prototype instance, whose slot-values are undefined is added for the classification process (fig. 6).

In the classification process the user can enter the value for any slot of the prototype-instance, trigger forward inferences or a constraint-satisfaction process to complete the information and compare the current description of the prototype with the known solutions.

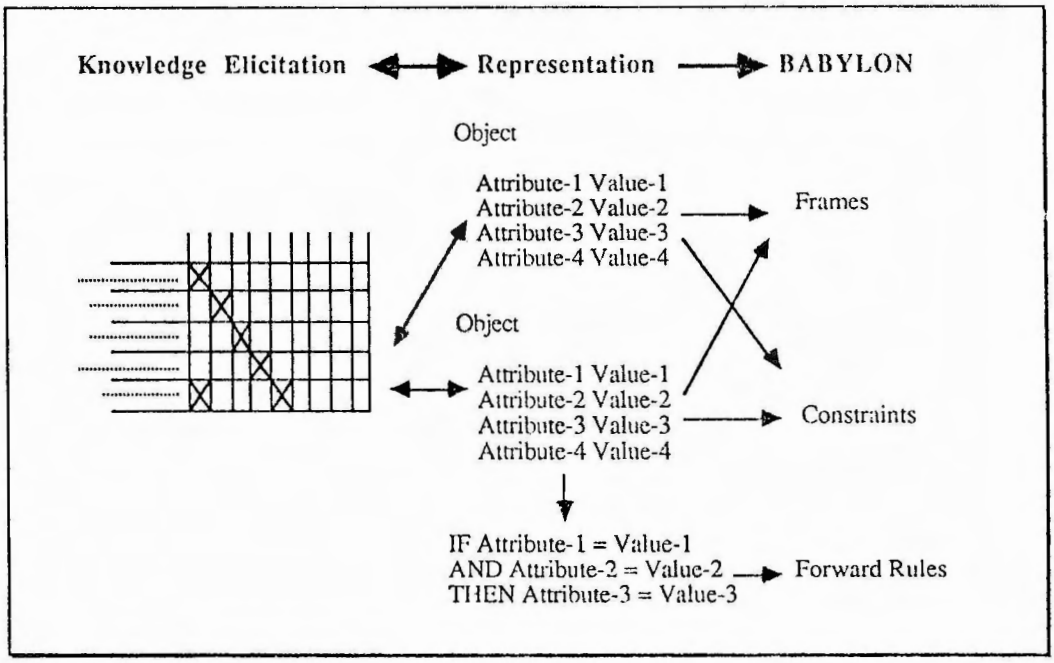


Fig. 7: An overview of the features of IRA-grid

6.4. Discussing IRA-grid

IRA-grid integrates one knowledge acquisition feature with several possible operationalizations of the knowledge, i.e. rules for forward chaining and constraints for consistent completion. These representations are used in a simple classification-like inference scheme.

The multiple operational representations of the same knowledge, i.e. the rules and constraints that are deduced from the descriptions, are not the goal of IRA-grid. The system aims at acquiring one kind of knowledge needed for the task of classification: a reach description of many concepts sharing comparable descriptions. It acquires knowledge-bits (descriptions of concepts, attributes, ranges etc.) and creates several operational representations of the knowledge to make overt what was really acquired. The transformation of the description of the concepts into rules and constraints aims only at confronting the user with multiple facets of his own knowledge, and thus triggering new insights that lead to extensions and revisions of the knowledge-base.

In IRA-grid we encountered several problems when we tried to confront the user of the system with the alternative facets of his knowledge, i.e. with the implications and constraints:

- If few concepts are described by many attributes with many different values many implications are true, and most implications are true for only one concept. The only means we have, to reduce the number of implications presented to the user, is by syntactically restricting the left and right sides of the rules representing the implications, as at this point of the knowledge-acquisition process we don't have enough knowledge yet to use model-based approaches to guide the implication analysis;
- Constraints are represented by relations between attribute-values. They are very hard to comprehend, and we have not yet found a way to present them in such a way that they lead to new insights and trigger modifications of the knowledge-base.

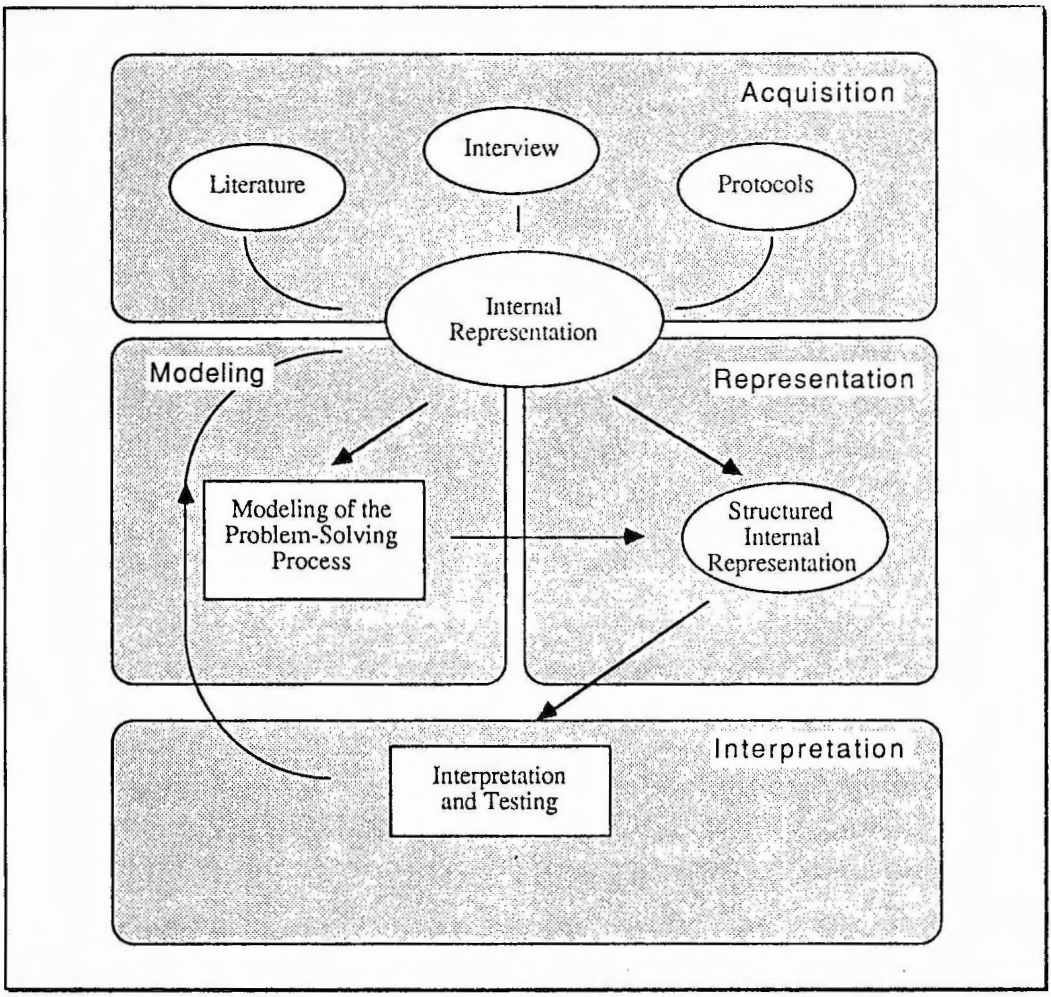


Fig. 8: The structure of IRMA, including several IRA-type acquisition tools, knowledge-level modeling facilities and an interpretation environment.

7. TOWARDS IRMA : COMBINING ACQUISITION, MODELING, REPRESENTATION AND INFERENCE

It is our long-term goal to combine knowledge-modeling, i.e. knowledge-level and symbol-level representation with acquisition techniques and with the corresponding inference techniques (see fig.8) in one environment called IRMA (Interpretation, Representation, Modeling and Acquisition). For that purpose we must match acquisition techniques (e.g. grids, forms, clusters, protocol-analysis, laddering) with inference- and task-building-blocks (e.g. data abstraction, heuristic match, classification, determine best solution, best match). This means that once we have an inference- or task-structure telling us which task-building-blocks are used, and how they are combined, then we can trigger the corresponding acquisition tools in IRA-* (IRA-grid, IRA-protocol, IRA-cluster or IRA-form) to fill in the symbol-level knowledge. The acquired knowledge-bits are combined to operational knowledge that performs the task they were acquired for.

Future research will be concerned with the descriptions of the knowledge-needs of problem-solving methods encountered in knowledge-based systems, and with the selection of the corresponding knowledge-acquisition tools.

8. SUMMARY

We presented the need for explicit knowledge-modeling in a task-independent knowledge acquisition environment. These explicit knowledge-level descriptions can be used to guide symbol-level knowledge-acquisition procedures. IRA-grid is a tool for knowledge acquisition for classification tasks, that allows for immediate testing of the acquired knowledge. IRA-grid will, together with other IRA-*, specialized on the analysis of protocols, clusters ,forms etc. be one part of IRMA. IRMA will combine the features of IRA-* with the need to explicitly represent problem-solving procedures in task- or inference structures.

REFERENCES

Bradshaw, J., Boose, J., Decision Analysis Techniques for Knowledge Acquisition : Combining Information and Preferences using Aquinas, to appear in IJMMS 1989

Breuker, J., Wielinga, B., van Someren, M., de Hoog, R., Schreiber, G., de Greef, P., Bredeweg, B., Wielemaker, J., Billaut, J.-P. Davoodi, M., Hayward, S.(1987), Model Driven Knowledge Acquisition, Memo 87-Esprit Project 1098, University of Amsterdam

Bylander, T., Chandrasekaran, B.(1986), Generic Tasks for Knowledge Based Reasoning: The right Level of Abstraction for Knowledge Acquisition, in : Boose, J.H., Gaines, B.R. (eds.), Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop 1986, Banff, Canada

Ericsson, K.A. & Simon, H.A. "Protocol Analysis.Verbal Reports as Data. In : The MIT Press, Cambridge, Mass. 1984

Di Primio, F., Wittur, K., BABYLON, a Meta-Interpretation Model for Handling Mixed Knowledge Representations, in : Proceedings of the 7th International Workshop Expert Systems and Their Applications, Avignon 1987, Vol. 1 821-833

Diederich ,J. , I. Ruhmann, M. May , A Knowledge Acquisition Tool for Expert Systems, in : International Journal of Man-Machine Studies (1987) 26, London, United Kingdom

Feigenbaum, E.(1980), Knowledge Engineering : The applied side of AI, Stanford Heuristic Programming Project Report P-80-31, Stanford University, Stanford

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

Gruber, T., A Method for Acquiring Strategic Knowledge, In: Boose, J.H. and Gaines, B. (Eds.): Proceedings of the "Knowledge Acquisition for Knowledge-Based Systems Workshop", Banff, Canada 1988

Güsgen, H-W., Junker, U., Voss, A., Constraints in a Hybrid Knowledge Representation System, in : Proceedings of IJCAI 1987, Vol 1. 30-33

Johnson, N., Tomlinson, Knowledge Representation for Knowledge Elicitation, In: Boose, J.H. and Gaines, B. (Eds.): Proceedings of the "Knowledge Acquisition for Knowledge-Based Systems Workshop", Banff, Canada 1988

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

Linster, M., A Critical Look at KRITON: Moving from First to Second Generation, In: Boose, J.H. and Gaines, B. (Eds.): Proceedings of the "Knowledge Acquisition for Knowledge-Based Systems Workshop", Banff, Canada 1988

McDermott, J.(1989) How Knowledge acquisition could be easy, Position paper for the workshop on foundations of expert systems,Ponte de Lima

Musen, M.A.(1988b), Conceptual Concepts of Interactive Knowledge Acquisition Tools, in : Boose, J.H., Gaines, B.R., Linster, M. (eds), Proceedings of the European Knowledge Acquisition Workshop 88, St. Augustin : GMD

Newell, A. (1982), The Knowledge Level, Artificial Intelligence (2), 87-127

S M A C : A PROTOTYPE OF A KNOWLEDGE MODELING AND ACQUISITION TOOL FOR THE EXPERT

Sandrine MOULIN
CAP SESA REGIONS
Agence Industrie
1, chemin du Pigeonnier de la Cépiere
31036 TOULOUSE CEDEX

ABSTRACT

Development of expert systems has brought to the fore the importance of knowledge acquisition. Without mastery of knowledge acquisition, it is impossible to build efficient expert systems. This paper discusses current knowledge acquisition methods, manual techniques as well as computer-based tools. Conclusions drawn from this review of concise state of the art, show the need for developping a new kind of tool. This paper presents the basic hypothesis of our work, the formalism used to create a tool close to the expert's model and the prototype we are developing. In conclusion, future extensions of the tool are discussed.

KEY WORDS

Knowledge acquisition, expertise transfer, cognitive model, object representation, hypermedia.

INTRODUCTION

Designers started to loose their enthusiasm for expert systems when they realized the difficulty of the knowledge acquisition (KA) phase. Since the 70s, knowledge acquisition is considered as one of the fundamental problems.

To solve this problem knowledge engineering has been created as a practical branch of artificial intelligence (AI). The knowledge engineer is a mediator between the expert and the computer. Knowledge acquisition methods have been progressively developed to provide a framework to knowledge engineers. Moreover, since 1982 research efforts [DAV 82] have been undertaken for developing KA tools directly usable by domain experts. Anyway, designing of an expert system is much related to "Art than to science" [BON 84].

I- STATE OF THE ART AND MOTIVATIONS

There are two kinds of knowledge acquisition methods:

- non computerized methods used by knowledge engineers: interviews, brainwriting, protocol analysis.
- computerized methods : tools for knowledge engineers or experts.

Methods using a knowledge engineer are more or less formalized. Attempts have been made such as the definition of repertory grids for the interviewer [LAF 86], brainwriting [WAR 71], and protocol analysis [NEW 72], but it is obvious now that the "choice of a knowledge engineer has a decisive importance for the success of the project" [HAR 88]. Indeed, the knowledge engineer must be skilled for communication and human relations in order to succeed in his/her job [HAR88]. Moreover acquiring knowledge through a knowledge engineer introduces a bias: the knowledge engineer may impose, unconsciously or not, his own understanding of the problem.

Thus research efforts are being made to develop tools for improving the knowledge acquisition process. Most of the tools used by knowledge engineers help them to structure the collected knowledge. For instance, KRITON [LIN 88] helps a knowledge engineer during the elicitation phase; resulting knowledge bases do not include any (meta)knowledge about the use of knowledge to solve a problem. KREME [ABR 86] is a knowledge representation editing and modeling environment, special for the development of large knowledge bases.

In order to eliminate the bias introduced by the knowledge engineer, tools have been developed to help the expert to transfer his knowledge to the computer. KNACK [KLI 86] is limited to the particular domain of expert systems evaluating electro-mechanical designs. TEREISIAS [DAV 82] helps the domain expert to write rules, but it is restricted to the inference engine of MYCIN. Other systems like MORE [KAH 85], MOLE [ESH 86], SALT [MAR 86], AQUINAS [BOO 86] have been developed. AQUINAS built from a psychological theory ("Personal construct theory" of Kelly [KEL 55]) is based on the notion of rating grids. It is essentially used for diagnosis.

Another approach is machine learning. Machine learning systems try to "increase their knowledge progressively by experimentation" [KOD 86]. Machine learning acquires concepts and structured knowledge. The two ways of doing it are similarity-based learning (SBL) - in which the system looks for similarities in a set of examples and dissimilarities between examples and counter-examples- and explanation-based learning (EBL) : the system learns from explanations gathered from the analysis of an example or a counter-example of the concept or rule to be learned. DISCIPLE [KOD 88] merges several types of learning according to the type of knowledge to be acquired.

Most of these tools are still prototypes. Moreover, domain experts find them difficult to use or they deal with a specific domain. As for machine learning systems, we think that interaction between the expert and the machine is still a better way to learn.

II- OUR APPROACH

If the knowledge acquisition phase is so difficult it is because experts do not know how to formalize their knowledge and know-how, because a large part of their knowledge has become "common sense" [MIN 88]. This is what the creation of expert systems brought to the fore.

That is why we decided to study an expert-oriented tool. The aim of the present work is not to design an expert system shell. Our goal is to provide the expert with a tool which will help him/her to structure and organize his/her knowledge and know-how for him/herself; so that, the expert will be able to analyze what he/she is doing. He/she will then be able to explain better his/her reasoning because he/she already thought about it.

Here are our hypotheses :

Hypotheses about a man at work [MAC 87]

Hypothesis 1 : "for any type of work, any human and his/her working tool form a living cell which therefore evolves with the time. This evolution produces an experience which gives him/her a better understanding of actions to come."

Hypothesis 2 : "any human helped by a dynamic tool of memorization of his/her own experience increases his/her intuitive and analytic senses."

Corollaries:

Corollary 1 : "any human is able to analyze his/her actions and to draw conclusions from it."

Corollary 2 : "any human who can overcome and explain all his/her actions is able to find others."

We want to design a easy to use tool because we think that the more simple it is, the more powerful it is. The tool is usable by any expert; as soon as the expert has memorized some knowledge in SMAC, it becomes his/her own tool; the expert don't have to explain his/her words; he/she directly interacts with the system in his/her formulations. We try to design a tool as general as possible, not dedicated to a single domain or restricted to a domain: for example, there are knowledge acquisition systems restricted to domains where solutions can be enumerated (analytic problems such as classification and diagnosis) or domains where the solutions are known (and not built up from components).

The basic idea is to bring an help to the expert in his work by memorizing and using his/her know-how. These two functions need to be done interactively and cyclically. The way the tool processes and lays out his/her knowledge and know-how (global or detailed way) will allow him/her to express others. We want to design a tool that will allow:

- memorization: we need to know how to express and structure our knowledge and actions on the domain.
- re-use: we need to know how to express the possible knowledge processings (deductions, inductions, links...)
- increase: the expert increases the collected knowledge when he/she both expresses and uses his/her knowledge.

It is only later that the gathered knowledge can be used to create a knowledge base if the expert wants it.

III- EXPERT AND PROBLEM-SOLVING

In order to guide the expert when he/she expresses his/her knowledge, a model of the expert has to be used. This model is based on hypotheses related to the types of expert's knowledge, its organization and the expert way of thinking.

III-1- THE EXPERT WAY OF ORGANIZING KNOWLEDGE

Various expert expressed knowledge and behavior elicited from interviews we have done [FER 88], [MOU 88] and from various studies [VOG 86], [VOG 88] pointed out the following features :

- the expert identifies and manipulates objects related to a physical reality and a location in the reference world. In a given context, each object is related to a given behavior, close to an "intentionality". An event is always signed. Every object manipulated by the expert has a label. The expert generates a micro-language which corresponds to the expertise area. We can study its localization, quantity and quality.
- the expert structures his/her knowledge with classifications; the more complete the expertise is, the deeper, more symmetrical and more homogeneous the classification is.
- the expert's approach is "positive": empirical and inductive. He/she uses the practical empiricism of his/her profession to search for rational explanations. The expertise is an extension, a continuation of the common sense.

III-2- EXPERT APPROACH TO SOLVE PROBLEMS

Theoretical studies have been done in order to understand the expert's approach when he/she is solving a problem. These studies rely on a top-down analysis of the expert's activity. Designing and producing something, is going from the definition of a

general aim to the plan and performance of series of elementary actions which lead to the product itself. The general aim is split up in less abstract aims, themselves split up once again, and so on [SIM 74], [SIM 84a], [SIM 84b].

But when research efforts deal with the observation and experimentation of the real experts' activity, the basic characteristic of the experts' planning is to be "opportunistic" [BIS 87]. Elementary steps of the process are activated by information noticed by the expert in the present state of his problem. These information suggest him decisions to be taken at any level in the hierarchy of aims.

This gives the experts - what we call - a very rich situational knowledge, and so they are able to react quickly when they face a new situation. Conversely, beginners have a very important analytic knowledge. The difference between situational and analytic knowledge has been pointed out by [BOY 87] [BOY 88]. The concept of situation is also encountered in other authors' books : "Isolated thinking is impossible, because the reasoning depends on a prior organization of a whole system of concepts, percepts, classes, categories - call it however you like- from which we understand all situations"[HOF 87]. "When he/she has to interpret a phenomenon, the expert tries to quickly identify a context, a situation. The expert has a precise perception of his/her data because he/she relates them with situations that he/she learned to identify" [VOG 88].

We have to get closer to the expert model if we want to design a tool which can be easily used. This way of facing the problem is one of the main characteristic of AI : an AI oriented project is able to "use or develop models or tools that can respect natural expression of problems"[VOG 88]. To get closer to expert model is to provide him/her a flexible context so that he/she can express his/her static knowledge with objects and dynamic knowledge with problems to be solved, associated with one or more situations.

According to our wish to provide a tool for experts, we can not think of giving him/her an object oriented language on which he/she can rely to describe his/her objects. It is necessary to guide him/her when memorizing his/her knowledge. We first present some concepts of object oriented languages (notion of classes with properties and methods as well as different hierarchies between these classes) then the concepts which we have added (rules, formulas and problems). We will finish up by the way this is proposed to the expert.

IV- THE USED FORMALISM

IV-1- OBJECT CONCEPTS

A class is defined by an identifier (its name) and a group of properties. A class is considered as a type, describing the structure of its "instances" called objects. More information can be associated to a class thanks to properties. A property is defined by its name, a domain and a default value. Properties of a class have all different names but these are local names. It means that properties of different classes can have the same name. A property includes a group of predefined constants as domain, for example integers or reals or character strings, defined in intension (reals) or in extension (enumeration).

The organization of the classes :

the aggregation : it is an abstraction with which you can consider that the relation between two objects is a higher-level object. This abstraction is very useful in CAD in which a lot of complex objects and hierarchical links are used.

the specialization/generalization : the specialization notion was first defined with the

first representations based on frames[MIN 75]. With this concept, we can define new classes from more general classes, just by defining the differences between this class and the one we inherit.

the association : it allows to formalize links between elements belonging to the same class or not.

IV-2- ADDITIONAL CONCEPTS

Collected objects as well as links between them are part of the expertise. But it is necessary to give the expert the possibility to describe a higher level in the expertise. That is why we introduced the notion of formulas and rules. We define three types of operators for the properties set:

- arithmetical operators : +, -, *, /
- relational operators : =, <, >, <=, >=
- logical operators : not, and, or

We can define in the properties set P

- a calculus: $Calculus_P$::=property P/arithmetical operator (list of $Calculus_P$)
- a formula: $Formula_P$ ::= relational operator (list of $Calculus_P$)
- a rule: $Rule_P$ ::= logical operator (list of $Formula_P$)

For every property of a class the user is able to write either formulas or rules. The links between the different classes will be exploited in this way -for the formulas and rules writing-: a class will have access in a recursive way to:

- its components properties
- its upper-class properties
- all its associated classes properties.

A formula or rule associated with a class needs at least one property of the class. So we get an universe U. This is a finite directed graph in which nodes are classes and edges are links.

IV-3 METHOD AND PROBLEM CONCEPTS :

The concepts described above allow to express static knowledge. In order to help the expert to express his/her way of solving problems, it is necessary to give him/her another context in which he/she will be able to describe how to use static knowledge.

Methods and problems concepts have been introduced in the system for that purpose. A method is different from a problem because it is "unitary", i.e. it applies to one and only one class. Therefore it is tied up with a class. A method is defined by a name, an input condition and an outpout. A text is associated with the method. It includes user explanation of what the method does. We define the input condition with the properties (of the class to which it is related and of upper-classes) that are necessary to apply a method. The output is define with the properties modified by the application of the method.

Conversely, to solve a problem, it is necessary to work with several classes. We define a problem by :

- a name
- a type
- a subject
- constraints
- one or more situations
- one or more ways to solve it

The concept of type is interesting for defining "categories of problem".

The subject of a problem is the name of a class. It is the focus of attention. We will use it to classify the problems described by the expert.

The constraints are expressed with the properties of the objects linked with the subject of the current problem. The usable operators are arithmetical and relational; "and" links are used to bind constraints.

Several situations/solutions are associated with a problem : the concept of situation corresponds to the notion of "entry point" in a problem. A problem has several "entry points": according to the state of design, a problem will be differently characterized: in a more or less precise way, depending on the availability of information.

Each solution tied up with a situation is composed of either the name of sub-aims to solve, or actions on the objects (methods), or different actions as showing a drawing, running an external program, and so on. The possible links between the elements of a solution are the orerators "and", "or", and "then".

We can model a top-down or a bottom-up approach with the linking between problems and sub-problems. Links are described in the solving technique so that the two approaches exist.

V- S M A C : A PROTOTYPE

SMAC has been developed using an HyperMedia : HyperCard TM [BRA 88] [GOO 87] [RIN 88] on Macintosh TM. The use of HyperMedia is appropriated for knowledge acquisition [GAI 88]. Their approach of knowledge is different from the one of expert systems, because they consider that the knowledge has a limited structure and processibility. They add a limited structure through the links intended to help the user rather than to further computer-based inferencing. HyperMedias facilitate the implementation of current concepts of knowledge bases, without requiring the informations system to be able to represent and process knowledge encoded in the media used (drawings, texts, films...). But, if the links are labeled, processes can be associated with links. Therefore, HyperMedia are a well suited media for memorizing human knowledge and relating it to other existing media. Moreover, the use of an HyperMedia such as HyperCard allows to generate easily nice users interfaces.

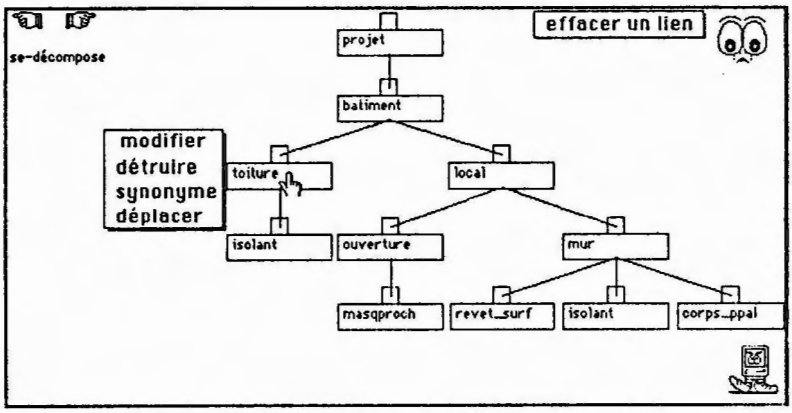
V-1- KNOWLEDGE MEMORIZATION

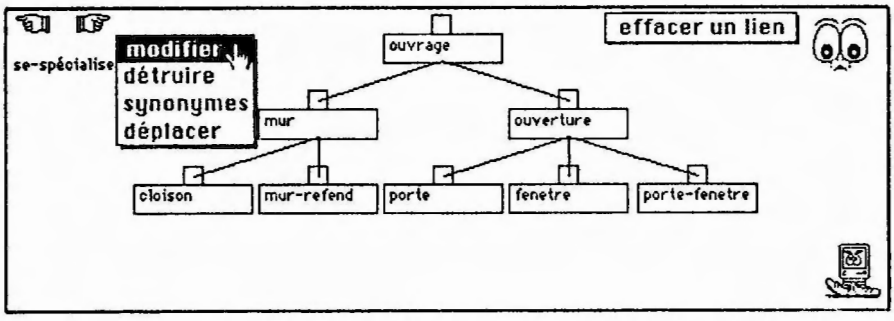
Memorizing the expert's knowledge means memorizing both domain knowledge and problem-solving knowledge. SMAC is divided in two parts : the first for the expert's universe acquisition, in terms of objects and links between these objects. The other to acquire the expert's problem solving method. The first part is necessary as basis for the second one; the expert will be able to describe his problem-solving method by relying on different objects composing his universe.

SMAC begins with a graphic capture of the different classes manipulated by the expert (for instance : project, building, apartment, wall...) with two possible structures : a tree of specialization and a tree of aggregations : as the expert manipulates objects, this approach is near the expert' model. Moreover, the experts often express their knowledge through classifications.

The first screen includes of empty "cells" in which the expert enters his/her objects and links them together. When a cell is full, the expert is able to modify it, delete it or define "synonyms" for this cell. The expert is provided with an easy to use Hypercard interface. When a modification, has been done on the screen, it is propagated through the whole system.

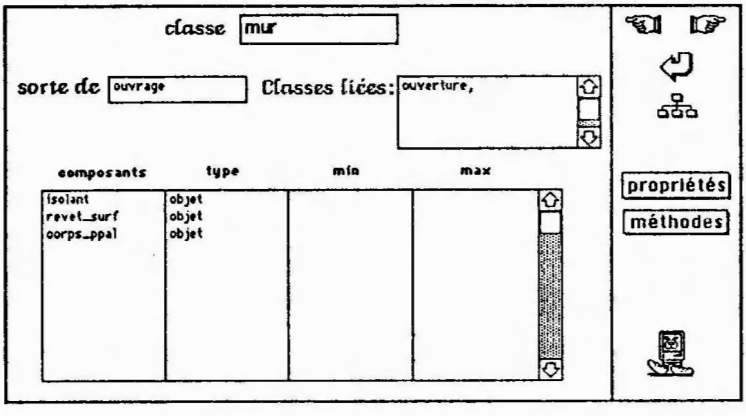
The application domain is house-building and thermics, in which CAOMIP has various activities.



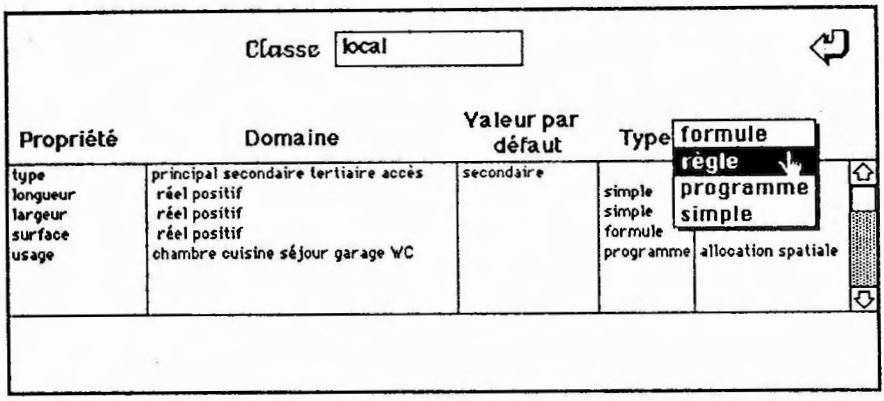


SMAC has then the names of the classes and of some links between these classes : it is the basic structuration. SMAC generates the classes (a class is then represented as an autonomous entity) with their components and upper-class. The expert can add properties to each class. He can also write formulas and rules with the properties of the class, properties of the upper-class, properties of the components classes or properties of other classes joined by a different link from "part-of" or "kind-of". Links intra-classes or inter-classes may be created. The writing of formulas and rules is made with the objects and properties already defined. The expert can also add methods associated with each class.

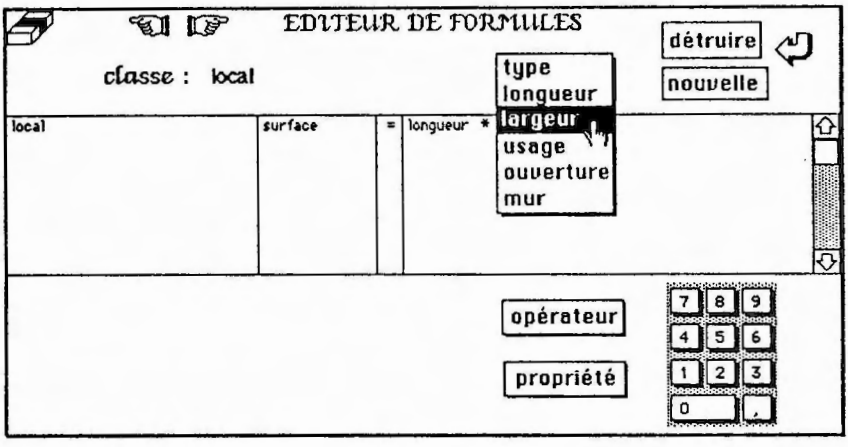
Example: the class "mur" with its components, upper-class and associated class



Example : the properties of the class "local"



An example of formula :



This incremental development (the expert is just able to use objects or properties already defined) prevents from capturing properties or objects non-memorized before.

Let us assume that the expert' knowledge is coherent; if there is a problem of coherence in a knowledge base, it is because the knowledge representation is too far from the expert' model. The coherence problems are consequences of knowledge representation.

Conversely, if the tool is close to the expert' model, it is just necessary to guide him/her.

Once the expert has described his/her objects and links, he/she has a design universe, on which he/she will rely to describe the problems and sub-problems he encountered.

The expert defines a problem by giving it a name and a type (optional). He chooses among the already defined classes, one class which is the subject of the problem.

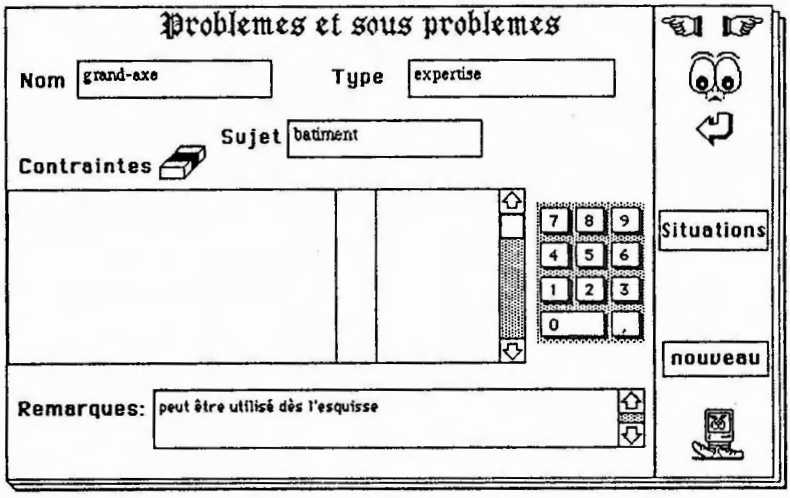
Then the expert describes the situations associated with the problem; the number of situations is not limited; a solving solution is tied up with each situation.

Defining the situations is defining the entry points in a problem. Thus the expert identifies the objects or the parts of objects which characterize a situation for a given problem. The link between the elements of a situation is "and".

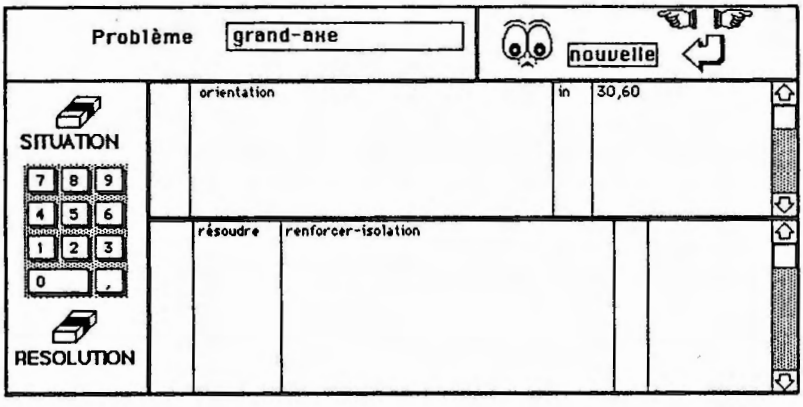
The solving technique is to give either the names of programs to run, the names of sub-problems to solve, the names of methods to apply or to express an expertise. The expert is not obliged to follow a hierarchical top-down approach. He can speak of a sub-problem before memorizing the situations/solutions associated (top down approach). But he can also follow a bottom up approach.

Nevertheless, we will have, at the end, a hierarchy of problems and sub-problems.

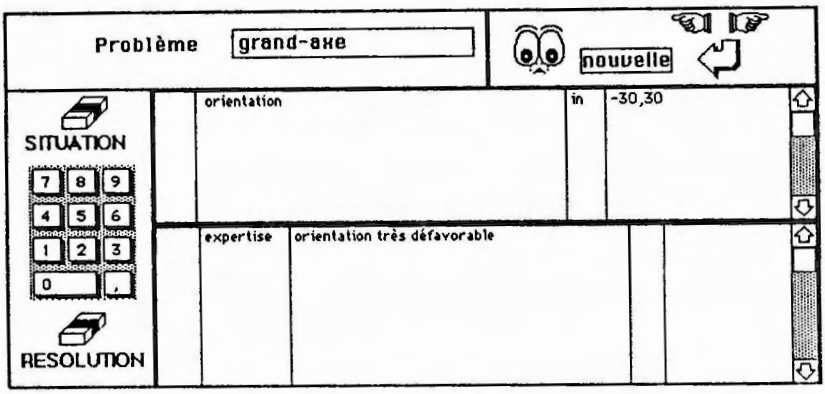
Examples : a problem and two of its situations/resolution associated



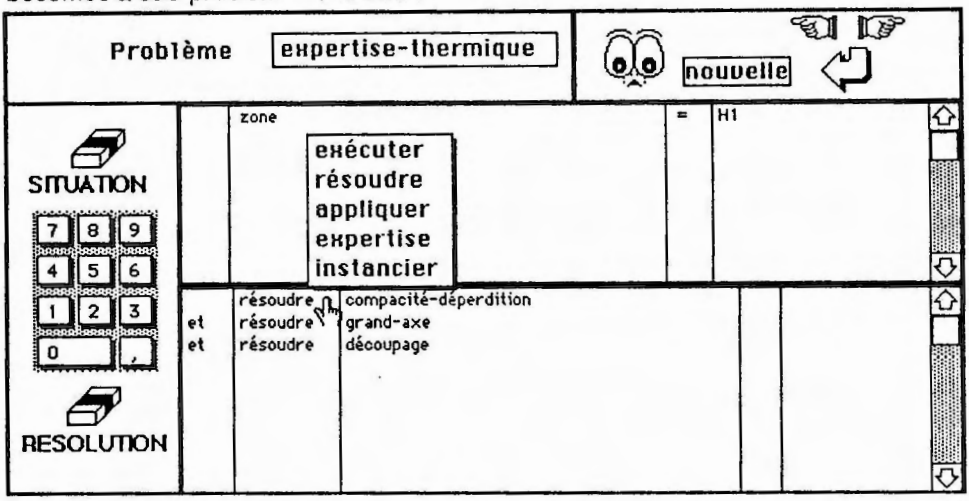
To solve the problem of "major-axis", when the building orientation is in 30,60 is to solve the sub-problem "to strengthen the insulation material":



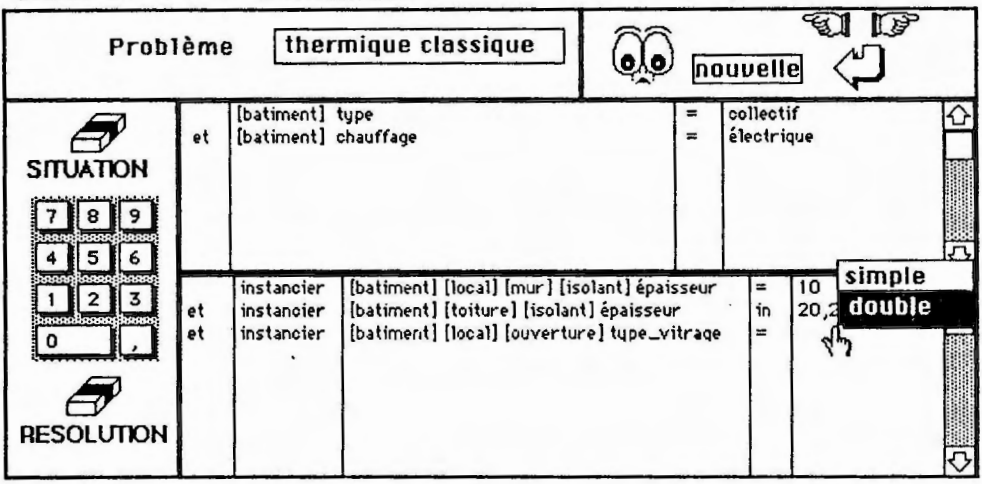
When the building orientation is in -30,30 , the orientation is too bad. The user needs to improve the orientation by modifying the situation. Nothing else has to be done:



Another problem : this a thermic expertise one; the previous problem "major-axis" becomes a sub-problem of this one :



Another one : to solve a classic thermic problem , the expert gives a value to the thickness of the insulation materials :



V-2 USING KNOWLEDGE

Two principal kinds of use are envisaged :

- using knowledge as "learning-memory"
- using knowledge as "opportunist design"

In the first kind of use, the expert uses the tool as a memory. The tool will "supply the expert's mind with fragments of ideas already used for similar works." [MIN 88] The expert will be able to consult his/her memory either from a problem already solved (how did I solve this problem the last time?), or from a situation (how did I face this situation?). The expert will become "in a way an old version of himself" [MIN 88].

In the second kind of use, the expert is solving a problem and the tool will help him/her to find out a solution among the ones it already knows. From a situation, the expert can try to come down to a similar situation. He can also try to recognize situations he already solved. The tool, thanks to memorized situations, can help him to recognize situations. It is an opportunist design approach because elementary steps are activated by informations noticed by the expert (bottom-up method). Once the situation is recognized, the solution can be achieved in a top-down way.

V-3- INCREASING KNOWLEDGE

The knowledge gathered in SMAC can be increased and improved in an incremental way during its use. The learning is done by interaction between the expert and the computer. Facing the behaviour of the system, the experts reacts, modifying the memorized knowledge and know-how.

Another possibility to increase collected knowledge, would be to give the expert the possibility to make analogies. To do that, SMAC needs to allow the definition of synonyms for each class : the name of the class, its properties/domain/default value and methods. SMAC would then generate automatically the different types of problems by using the synonyms.

CONCLUSION

The approach consisting of identifying first the classes manipulated by the expert and then the problems he has to solve, suits well to a modular knowledge acquisition, and this is for many reasons: on the one hand, autonomous classes are clearly identified and the links between classes are identified. On the other hand, problems are delimited that is to say, that we distinguish links between problems and links between the classes which have generated the problem. The use of classes and problems also allows a step by step approach, in which classes are introduced and completed as the needs appear. So, classes and problems become real memories of the expert's know-how.

The prototype allows the memorization of the expert's universe as well as part of the description of his problem-solving approach. The part concerning the use and increase of knowledge is underdevelopment.

Such a tool will allow memorization and use of the expert' knowledge and know-how using different forms : the list of problems that SMAC can solve, the actions to do to solve a given problem, the objects necessary for solving a problem, the list of situations that SMAC can recognize, the sub-problems of a problem, and so on. Moreover, if the definition of synonyms is provided at whatever level of the expert' universe, it will be possible to solve an analogous problem.

Finally, we are aware of the difficulty of experimentation because it needs a lot of

availability from the expert. Experimentations we made so far are mainly focused on model appropriateness [MOU 88]. But in the next months, we will have the possibility to work with two experts and we will test the tool on its whole.

REFERENCES

[ABR 86] ABRETT G. and BURSTEIN M.H., "The Kreme knowledge editing environment", Knowledge acquisition for knowledge-based systems workshop, Banff, Canada, novembre 1986.

[BIS 87] BISSERET A., " Les activités de conception et leur assistance " INRIA-Rocquencourt. Projet de psychologie ergonomique. Bulletin de liaison de la recherche en informatique et automatique n°115/1987.

[BON 84] BONNET A., "L'intelligence artificielle : promesses et réalités" InterEditions, 1984.

[BOO 86] BOOSE J.H and BRADSHAW J.M. "Expertise transfer and complex problems using Aquinas as a knowledge acquisition workbench for expert systems", Knowledge acquisition for knowledge-based systems workshop, Banff, Canada, novembre 1986.

[BOY 87] BOY G., " Représentation Analytique et Représentation Situationnelle " Séminaire ENAC-Erreurs humaines et Automatisation- mars 1987

[BOY 88] BOY G., FALLER B. and SALLANTIN J., "Acquisition et ratification de connaissances", Actes des journées PRC-GRECO, 1988

[DAV 82] DAVIS R. and LENAT D.B.,Knowledge-Based Systems in Artificial Intelligence, New-York : McGraw-Hill, 1982.

[ESH 86] ESHELMAN L., EHRET D., McDERMOTT J. and TAN M., "Mole : a tenacious knowledge acquisition tool", Knowledge acquisition for knowledge-based systems workshop, Banff, Canada, novembre 1986.

[FER 88] FERRIES B. and MOULIN S., " L'apport des systèmes experts pour la conception thermique du projet architectural dans le système Krèpis " - Intelligence Artificielle et CAO en BTP; recherche et applications-CIMA- HERMES- fév 88

[GAI 88] GAINES B.R., "Second generation knowledge acquisition systems", Proceedings of the European Knowledge Acquisition Workshop - EKAW, Bonn, juin 1988.

[HAR 88] HART A., " Acquisition du savoir pour les systèmes experts " Sciences cognitives MASSON - 1988

[GOO 87] GOODMAN D., " The complete HyperCard handbook " Bantam computer books -1987

[HOF 87] HOFSTADTER D. and DENNETT D., "Vues de l'esprit, fantaisies et réflexions sur l'être et l'âme", InterEditions, septembre 1987.

[KAH 85] KAHN G., NOWLAN S. and McDERMOTT J., "More : an intelligent knowledge acquisition tool", Proceedings of the International joint conference on artificial intelligence, IJCAI 1985.

[KEL 55] KELLY G.A., "The psychology of personal constructs", New York, Norton, 1955.

[KLI 86] KLINKER G., BENTOLILA J., GENETET S., GRIMES M. and McDERMOTT J., "Knack - Report driven knowledge acquisition", Knowledge acquisition for knowledge-based systems workshop, Banff, Canada, novembre 1986.

[KOD 86] KODRATOFF Y., "Leçons d'apprentissage symbolique automatique", Cepadues Editions, Techniques avancées de l'informatique, novembre 1986.

[KOD 88] KODRATOFF Y. and TECUCI G., "Learning at different levels of knowledge", Proceedings of the European Knowledge Acquisition Workshop - EKAW, Bonn, juin 1988.

[LAF 86] LAFRANCE M., "The knowledge acquisition grid : a method for training knowledge engineers", Knowledge acquisition for knowledge-based systems workshop, Banff, Canada, novembre 1986.

[LIN 88] LINSTER M., "Kriton : a knowledge elicitation tool for expert systems", Proceedings of the European Knowledge Acquisition Workshop- EKAW, Bonn, juin 1988.

[MAC 87] MACIAS F., CAOMIP, Toulouse, Private communication - 1987

[MAR 86] MARCUS S., "Taking backtracking with a grain of Salt", Knowledge acquisition for knowledge-based systems workshop, Banff, Canada, novembre 1986.

[MIN 75] MINSKY W., "A framework for representing knowledge" The psychology of computer vision - 1975

[MIN 88] MINSKY M., "La société de l'esprit", InterEditions, avril 1988.

[MOU 88] MOULIN S. and CARADANT D., "Une démarche expérimentale d'acquisition de connaissances à travers la réalisation du système d'aide à la conception architecturale Krèpis." Colloque IASTED systèmes experts et applications - Genève - 1988

[NEW 72] NEWELL A. and SIMON H.A., "Human problem solving", Prentice Hall, Englewood Cliffs, 1972.

[RIN 88] RINALDI F., "HyperCard : la programmation en HyperTalk" Cedic- NATHAN 1988

[SIM 74] SIMON H.A., " La science des systèmes, science de l'artificiel " EPI-1974

[SIM 84a] SIMON H.A., " The sciences of Artificial " MIT Press - Third printing-1984

[SIM 84b] SIMON H.A., " L'unité des arts et des sciences : la psychologie de la pensée et de la découverte " AFCET/ INTERFACES N°15 -janvier 1984

[VOG 86] VOGEL C., " Systèmes experts : savoir-faire " Informatique Internationale - Vitrolles-1986

[VOG 88] VOGEL C., " Génie cognitif "Sciences cognitives MASSON 1988

[WAR 71] WARFIELD J.N., "Societal systems : planning, policy and complexity", John Wiley Editor, 1971.