# CURRENT TRENDS IN KNOWLEDGE ACQUISITION

BOB WIELINGA,
JOHN BOOSE
BRIAN GAINES,
GUUS SCHREIBER,
MAARTEN VAN SOMEREN,
EDITORS

FRONTIERS IN ARTIFICIAL INTELLIGENCE AND APPLICATIONS

IOS
Amsterdam, Washington, Tokyo

# Current Trends in Knowledge Acquisition

Edited by

**Bob Wielinga**
University of Amsterdam

**John Boose**
Boeing Computer Services, Seattle

**Brain Gaines**
University of Calgary

**Guus Schreiber**
University of Amsterdam

**Maarten van Someren**
University of Amsterdam

*IOS Press*

*1990*

Amsterdam • Washington, DC • Tokyo

# Preface

Knowledge acquisition has become a major area of artificial intelligence and cognitive science research over the past decade. The growing number of publications and workshops is a clear sign of this development. This book contains the contributions to the fourth European Knowledge Acquisition Workshop (EKAW-90) held in the Netherlands. The papers form the basis for discussions on issues in knowledge acquisition, and as such they reflect both the current state of the field and the opinions of the authors on important research topics. The papers in this book show that the area of knowledge acquisition for knowledge-based systems is still a diverse field in which a large number of research topics are being addressed. However, several main themes run through the papers.

**Knowledge integration** First, the issue of integrating knowledge from different sources and K.A. tools is a salient topic in many papers. Brazdil and Torgo explicitly address the problem and propose a method based on the use of empirical evidence as a criterion for integration. Shadbolt and Wielinga describe a scenario in which integration is presented as a knowledge-based activity. Another paper from the ACKnowledge project by Eggen, Lundteigen and Mehus, discusses integration from a computational viewpoint. Van Someren, Zheng and Post discuss how three types of knowledge obtained through different knowledge acquisition techniques can be combined in an integrated system.

Several papers deal with advanced workbenches in which different tools for knowledge acquisition are integrated. Gaines and Linster describe the integration of the tools from KSS0, based on the repertory grid approach, with the representation system Babylon through hypertext. They advocate an integration methodology in which existing tools are integrated within a common environment. The Keats system also combines several knowledge engineering tools in one environment although emphasis of Motta's paper is on methodology. The Shelley system (Anjewierden et.al.) achieves a large degree of integration of K.A. tools through the use of a repository of common persistent objects representing the results of various K.A. tools.

**Knowledge modelling** A second major topic in the papers is that of *knowledge modelling*. Recent research in knowledge-based systems emphasises the use of generic models of reasoning and its underlying knowledge. Such models are generally placed at the knowledge level. In spite of the growing consensus on the importance of knowledge models in knowledge acquisition, little has been done to create a unified framework for knowledge modelling. Karbach, Linster and Voss present a comparison of four influential approaches to knowledge modelling and its role in knowledge acquisition. Even if one disagrees with some of their

conclusions, the paper is an important contribution to the discussion of the nature of such models used in knowledge acquisition.

An important trend in the area of knowledge modelling aims at the formalisation of knowledge models. Wetter presents a first order logic representation of KADS models and compares it with earlier attempts to formalise KADS models by Wielinga et. al. Kowalczyk and Treur use methods of logic to construct formal specifications of generic tasks. Balder and Akkermans describe a tool which can support the structuring of formal specifications of KADS knowledge models.

**Techniques and tools for knowledge elicitation** Where the field of knowledge acquisition was without tools and techniques a decade ago, now there is a rapidly growing body of techniques and tools. Apart from the integrated workbenches already mentioned above, several papers in this book present new tools. Major and Reichelt describe a tool that supports knowledge elicitation through laddering. Boose, Shema and Bradshaw discuss the use of a variety of knowledge acquisition tools in order to capture design knowledge in a *corporate memory facility.*

Meyer, Booker and Bradshaw address the problem of *bias* in knowledge elicitation. They describe a six-step method for handling bias in a particular application. Hoppe approaches the important issue of validation of knowledge as a problem of acquiring the user intention and establishing whether the user intention meets the actual system. Davis and Bonnell present a knowledge acquisition method -*Conceptual Abstraction*- derived from semantic modelling for databases.

Two papers deal withy methods for acquiring knowledge from text. Schmidt and Schmalhofer describe a systematic method for knowledge acquisition from text in which the domain expert plays an important role. Ruqian and Cungen approach the problem of the acquisition of domain knowledge from books through the definition of an intermediate language which is close to the original natural language, but which can be automatically translated to a knowledge base. The translation from the original text to the intermediate language is done by the knowledge engineer.

**Knowledge acquisition and machine learning** Although knowledge acquisition and machine learning have been considered as separate subfields of AI, there is a recent tendency for the two fields to come together. The ACKnowledge project represented by the papers of Eggen et. al. and Shadbolt and Wielinga, combines machine learning techniques with more conventional knowledge elicitation techniques. Aamodt presents a framework in which reasoning, problem solving and learning together form a knowledge intensive system that can acquire knowledge from its own experience.

Induction has been shown in the past to be an effective means of knowledge acquisition by itself. Applying induction techniques usually requires considerable engineering of the data relies on a substantial amount of data. A major problem of applying induction techniques to knowledge acquisition problems is that the results of induction are often difficult to understand for expert, the end user of the knowledge base and also for the knowledge engineer. This makes it difficult to integrate induction with manual knowledge engineering

and causes problems with explanation for users. Van Someren et.al. describe these problems and propose a solution by decomposing the problem into subproblems that can adequately de dealt with by either induction or manual techniques. Kononenko shows that experts prefer naive bayesian statistics over decision trees and Nunez shows how extra knowledge such as costs, hierarchies of descriptors, etc. can be used to obtain more comprehensible results of induction.

We are sure that the reader will find in the material presented in this book stimulating new ideas and topics for further research in this challenging field.

## Acknowledgement

As editors we would like to thank the following members of the EKAW-90 programme committee for their assistance in reviewing and selecting the papers for this book.

Tom Addis, University of Reading
Guy Boy, Centre d'Etudes et de Rech. de Toulouse
Jeffrey Bradshaw, Boeing Computer Services
Jean Gabriel Ganascia, Université Paris-Sud
Yves Kodratoff, Université Paris-Sud
Marc Linster, GMD, St Augustin
John McDermott, Digital Equipment Corporation
Ryszard Michalski, George Mason University
Katharina Morik, GMD, St Augustin
Nigel Shadbolt, University of Nottingham
Mildred Shaw, University of Calgary

*The editors*
*May 1990*

# Contents

**A Computational Model of Knowledge-Intensive Learning and Problem Solving** ..... 1
A. Aamodt

**Multiple Knowledge Acquisition Strategies in MOLTKE** ..... 21
K.-D. Althoff, F. Maurer and R. Rehbold

**Shelley – Computer Aided Knowledge Engineering** ..... 41
A. Anjewierden, J. Wielemaker and C. Toussaint

**StrucTool: Supporting Formal Specifications of Knowledge-Level Models** ..... 60
J. Balder and H. Akkermans

**Capturing Design Knowledge for Engineering Trade Studies** ..... 78
J.H. Boose, D.B. Shema and J.M. Bradshaw

**Knowledge Acquisition via Knowledge Integration** ..... 90
P.B. Brazdil and L. Torgo

**Producing Visually-based Knowledge Specifications for Acquiring Organizational Knowledge** ..... 105
J.P. Davis and R.D. Bonnell

**Integration of Knowledge from Different Knowledge Acquisition Tools** ..... 123
J. Eggen, A.M. Lundteigen and M. Mehus

**Development of Second Generation Knowledge Acquisition Systems** ..... 143
B. Gaines and M. Linster

**Validation of User Intention** ..... 161
T. Hoppe

**Model-Based Approaches: one label – one idea?** ..... 173
W. Karbach, M. Linster and A. Voss

**Comparison of Inductive and Naive Bayesian Learning Approaches to Automatic Knowledge Acquisition** ..... 190
I. Kononenko

**On the Use of a Formalized Generic Task Model in Knowledge Acquisition** ..... 198
W. Kowalczyk and J. Treur

**ALTO: An Automated Laddering Tool** ..... **222**
N. Major and H. Reichgelt

**A Flexible Six-Step Program for Defining and Handling Bias in Knowledge Elicitation** ..... **237**
M.A. Meyer, J.M. Booker and J.M. Bradshaw

**Methodological Foundations of Keats, The Knowledge Engineers' Assistent** ..... **257**
E. Motta, T. Rajan, J. Domingue and M. Eisenstadt

**Decision Tree Induction using Domain Knowledge** ..... **276**
M. Nunez

**Towards Knowledge Acquisition from Domain Books** ..... **289**
L. Ruqian and C. Cungen

**Case-Oriented Knowledge Acquisition from Texts** ..... **302**
G. Schmidt and F. Schmalhofer

**Knowledge-Based Knowledge Acquisition: The next generation of support tools** ..... **313**
N. Shadbolt and B.J. Wielinga

**Cases, Models or Compiled Knowledge; a Comparative Analysis and Proposed Integration** ..... **339**
M. van Someren, L.L. Zheng and W. Post

**First Order Logic Foundation of the KADS Conceptual Model** ..... **356**
T. Wetter

# A Computational Model of Knowledge-Intensive Learning and Problem Solving

Agnar Aamodt

Knowledge Engineering Laboratory, ELAB-RUNIT, SINTEF
N-7034 Trondheim-NTH, Norway
and
Department of Electrical Engineering and Computer Science
University of Trondheim
agnar.aamodt@elab-runit.sintef.no

**Abstract.** If knowledge-based systems are to become more competent and robust in solving real world problems, they need to be able to adapt to an evolving domain and a changing environment. This paper proposes a computational model - a framework - for knowledge-intensive problem solving and learning from experience. The model has been instantiated in an architecture for knowledge-intensive case-based reasoning and learning called CREEK (Case-based Reasoning through Extensive Expert Knowledge). The importance of a thorough, extensive knowledge model to support the reasoning and learning processes is emphasized. In case-based reasoning a problem is solved by retrieving a similar past problem case, and using this case in solving the new problem. Learning becomes a process of extracting relevant information from a problem just solved, and integrating the new case into the existing case-base. The computational model presented combines case-based learning and reasoning with model-based and rule-based methods. The type of problems addressed are problems within open, weak theory domains.
*Keywords: Machine Learning, Knowledge modelling, Case-Based Reasoning.*

## 1. Introduction

### *1.1. General*

Knowledge acquisition for high quality, expert level, knowledge-based systems is a difficult and time consuming effort. Such systems are aimed at solving complex real world problems - like medical diagnosis, fault-finding and repair in industrial processes, design of technical production equipment, planning of financial investments. They may have to cope with problems on the border of (or slightly outside) their special subject domain. The systems will have to operate in a continually evolving environment, which requires frequent updating and refinement of a system's knowledge and problem solving ability. Ideally, a system should be able to update its knowledge after each problem solving session. The problem of *knowledge maintenance* is therefore a major challenge for the AI community. If a system is to continually maintain and improve its problem solving competence, methods for *automated learning from experience* is required. The ability to learn from each attempt to solve a problem during normal run-time operation will be referred to as *sustained learning*. We argue that if such a learning process shall be effective and useful, it should be based on - and focused by - a thorough understanding of the subject domain. That is, an underlying deep knowledge model should guide the learning of operational knowledge, as well as the problem solving that makes use of it.

People are good problem solvers because we base our methods on a general understanding of the world we operate in. We also maintain 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 tasks, we learn to do things better next time.

This paper describes a computational framework for knowledge-intensive learning and problem solving. The model combines *case-based reasoning* [28, 27, 11a, 12a] with methods that utilize generalized knowledge, i.e. deep models and heuristic rules. A system architecture, called CREEK, has been developed, based on the framework[1]. A comparison of the architecture with two other relevant approaches to knowledge-intensive case-based reasoning - the PROTOS [4, 5] and CASEY [14] systems - is made in [3], and a general overview of the approach with an example from modelling of a mud diagnosis system is given in [2].

### *1.2. Problem Solving in the Real World*

Most real-world problems that knowledge-based systems are addressing, are *open problems*. An open problem is characterized by having a weak or intractable domain theory. That is, there is no theory that completely describes the problem domain, and from which correct solutions can be derived or proved. Typical weak theory domains are medical diagnosis, geological interpretation, investment planning, and most engineering domains (i.e. domains that involve interaction with the external world). A weak domain theory is characterized by uncertain relationships between its concepts. The stronger the theory, the more certain are its relationships. At the very far end of this spectrum are the complete theories. The relationships of complete theories are certain, e.g. as expressed by standard logical or structural relations. Domains with strong domain theories are, for example, mathematical domains, 'block worlds' and some games - like checkers and chess. Even some strong theory domains may incorporate problems that turn out to be open when addressed by a problem solver. Chess, for example, has a strongest possible - a complete - domain theory, but solving the problem of chess-playing by an implementation of the theory is violated by its intractability. The concept of a 'winning plan' in chess is theoretically deducible, but there is no efficient algorithm to infer it in the general case.

Adding to the problems of weak or intractable theories, problem solving in the real world is, in general, complicated by incomplete and noisy problem specifications. This puts two important requirements on an intelligent problem solver:

- The ability to infer missing information, to generate and check expectations, and to justify whether a piece of information is relevant or not.
- The ability to interact with the information source (usually a human being) in order to extract additional information, and for confirmation or rejection of dubious conclusions.

Hence, a real world problem solver should be able to dynamically change the scope and goals of a problem during problem solving, and to deal with the contradictions and inconsistencies that are introduced in this process. The basic method that enables a system to infer meaningful results, justify its own decisions, and interact intelligibly with the user, may be viewed as a process of *hypothesis generation* followed by *producing and evaluating explanations*. In real world, weak theory domains, it is seldom the case that an inferred result can be *proved* true or false. Instead, the problem solver has to produce and justify its conclusion by explaining to itself why the conclusion is a plausible one (see, e.g., [32]). A result derived from a previous problem solving case, for example, may need to be supported by an explanation based on general domain knowledge. The emphasis on explanation in the reasoning process requires a thorough and deep knowledge model in order to produce plausible, meaningful explanations.

### *1.3. Sustained Learning*

A goal of machine learning research is to enable self-learning in computer systems. This goal is approached by developing theories and methods for processing and retaining observations (facts that are input to the system) in a way that enables them to be used in future problem solving. This is done either by inducing and saving generalised forms of the observed facts, or by keeping them as concrete cases from which solutions to similar problems may later be derived. Learning always imply some kind of generalisation. A fundamental difference in learning methods is whether the generalisation steps are performed when knowledge is stored (as in induction-based learning), or when problems are solved (as in case-based reasoning).

A fundamental principle for sustained learning is that each problem solving experience is a powerful source of learning. A necessary criteria for the learning to be successful is that a mistake once made will not be repeated. A role of the deeper model is to give the system a rich knowledge fundament for problem solving and learning: It will enable knowledge intensive case-based learning and reasoning by generating

[1]The development of CREEK is part of a Ph.D research - now in its terminal phase - supervised by Professor Arne Sølvberg.

explanations to justify reasoning steps, extracting relevant features, retrieving relevant cases, indexing cases and deciding which parts of a case to store and which previous cases to remove.

Figure 1 shows a basic scheme for sustained learning. The middle box illustrates - at an abstract level - the necessary steps: First, make an internal description of the problem, i.e. try to understand the problem by representing it within the system's internal knowledge model. Next, use whatever type of knowledge and reasoning method appropriate to produce a satisfactory solution to the problem. Finally, retain knowledge learned from this experience, so that the same or a similar problem may be solved with less user interaction, or more efficiently, in the future.

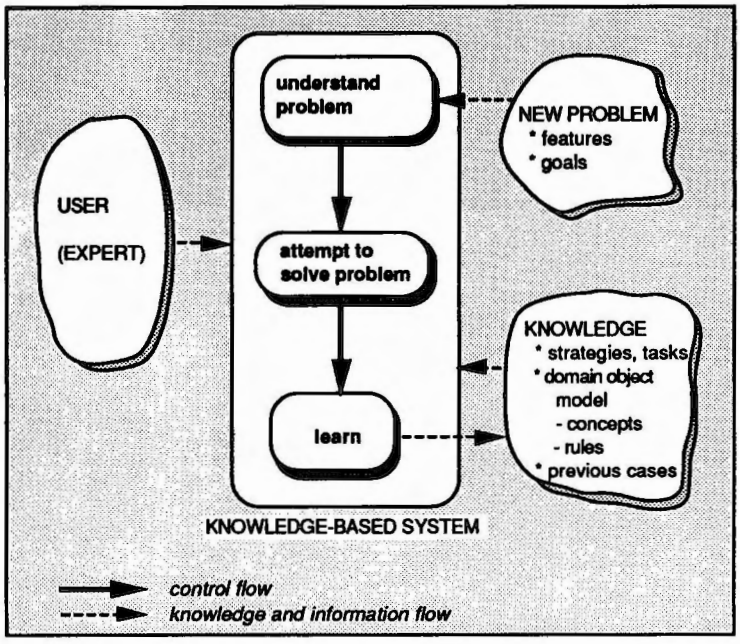



**Figure 1: Sustained learning through problem solving**
The figure shows a model of sustained learning. Knowledge - of different types - is used for learning as well as problem solving, and gets updated by the learning methods. The model incorporates user interaction both in problem solving and learning.

None of today's commercially available machine learning systems are able to learn continually from experience in real world environments; nor are classical induction methods like ID3 [27a], Version Space [22] and AQ [20]. Beside being unable to take advantage of an evolving knowledge base in their learning process, these methods 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 a concept. Such definitions may be adequate for mathematical and other strictly formal concepts, but as shown by several authors [41, 33, 31] it is extremely difficult to present classical definitions of naturally occurring concepts like 'game', 'bridge', 'chair', 'car', 'bird', etc. Smith & Medin [33] distinguish between the 'classical', the 'probabilistic' (degree of prototypicality) and the 'exemplar' (purely extensional) view of concept definitions. Natural concepts are defined intensionally by a set of *typical* properties - i.e. a set of default properties - and/or extensionally as the set of individes, or exemplars, that are classified by the concept. Methods that try to learn non-classical concept definitions need support from an extensive domain model in order to explain why an exemplar is within the range of a concept, why some features are more relevant than others, etc.

## 2. An integrated framework

This paper describes the CREEK framework for supporting the development and maintenance of competent systems that learn from experience. The three properties listed below should be satisfied by the resulting systems. Hence these system properties defines the requirements of the framework:

- An expressive and extendible knowledge representation formalism, enabling an explicit and thorough modelling of relevant knowledge types.
- Problem solving and reasoning methods that are able to effectively combine reasoning from past cases with reasoning within a competent and robust model of more general domain knowledge.
- A learning method that is able to retain concrete problem solving experiences, and integrate each experience into the knowledge model in a way that gradually improves knowledge quality and problem solving performance.

Although differing in its case-based approach, the CREEK approach has some similarities with other systems that take a knowledge modelling approach to machine learning, such as the BLIP [23] and DISCIPLE [35] systems. They all address learning and problem solving in weak theory domains, and include user interaction in their learning methods. Another common feature with CREEK and DISCIPLE is the emphasis on learning as a continous knowledge maintenance (knowledge refinement) process. But while DISCIPLE learns by generalizing an experience into a rule, CREEK learns by retaining the specific experience in a form that enables it to be used to solve similar problems later. IN addition to the case learning method, CREEK differs from both DISCIPLE and BLIP in its emphasis on a thorough, tightly coupled domain model - enabled by an expressive and flexible representation language, its explicit representation of control knowledge, and its combined reasoning methods.

### *2.1. Basic notions*

Unfortunately, there is no common consensus in the AI field about the meaning of basic terms. We will therefore start by describing the meaning and inter-relationships of some central terms frequently referred to in this paper, namely the terms *problem solving*, *learning*, *reasoning*, and *inference* The way the terms are used in this report implies a three-level process structure as shown below:

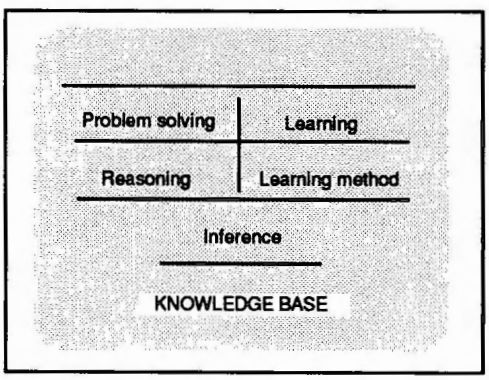



**Figure 2: Structure of knowledge based processes**

*Problem solving* is a process that takes a problem description, a goal, and a knowledge base as input, and derives a solution that satisfies the goal. The goal contains a specification of the requirements that must be fulfilled in order for a result to be a solution to the problem. A problem may be structured as sub-problems, and the problem solving process correspondingly split into sub-processes. The ability to solve problems is part of an agent's expertise, which includes domain knowledge, control strategies (plans), and reasoning methods.

*Learning* is a process that takes an existing problem solving system and new information as input, and derives a system where the quality and/or efficiency of problem solving is improved. To learn may, of course, be regarded as a problem in its own right, and problem solving could be said to subsume learning. To avoid confusion, problem solving will here solely refer to solving application problems.

*Reasoning* is a process that take some facts and a goal as input, and derives a result by applying one or more *inference* methods to a body of knowledge. Reasoning may be regarded as a more general term than problem solving, in the sense that its input and output may be any kind of information, not necessarily a problem description and a solution. Reasoning is also a sub-process of problem solving, since it is used to derive results that contribute to solving a problem. The term reasoning is most often used in connection with

problem solving, and characterizes parts of the problem solving process, as in *rule-based reasoning, model-based reasoning, case-based reasoning*. Being a general process of deriving a result through inference, reasoning is also involved in some learning processes - and particularly in knowledge-intensive learning. In the machine learning literature, however, terms like *learning method* or *learning algorithm* are more frequently used when talking about particular sub-processes of learning, reserving the term reasoning to the problem solving process. This is also the view adopted in our work.

*Inference* denotes the lowest level of processes in the hierarchy, i.e. the building blocks for reasoning and learning. The inference methods define the primitive operations on the knowledge, and hence, the basis for its semantic interpretation. There are three basic types of inference processes:

- Deduction, which is a truth preserving inference process where an inference step typically is a logical derivation of a theorem from a theory that is assumed to be both consistent and complete. Deduction is based on a universal inference rule, like *modus ponens*. As an inference rule of first order predicate logic, deduction guarantees the truth of the conclusion given the truth of the premisses. But the requirements on the knowledge as a non-contradictory set of theorems are often hard to meet in real world contexts.

- Abduction, also called *inference to the best explanation*, is a kind of reverse deduction that preserves falsity, not truth. An abductive conclusion therefore has to be supported by an explanation that leaves the particular conclusion as the best choice. Abduction is also called *plausible inference* ([Charniack-84]), and is closely tied to generation and evaluation of explanations. Abduction is considered to be the kind of inference typically made in diagnosis (at a top level, at least) where a set of symptoms infers a fault by being explained by the fault, i.e. by generating a plausible argument that supports a particular fault and weakens competitive fault hypotheses.

- Induction is a process that synthesizes a general description (e.g. a concept definition) from a set of examples and, possibly, counterexamples. Induction is similar to abduction in that the truth of the premisses (the examples) does not guarantee the truth of the conclusion (the generalization). Therefore, an inductive conclusion should also be supported by a justification.

Deductive inference is a strong method, but its applicability for reasoning is limited by assumptions that do not generally hold for real world situations: A world - typically assumed to be closed - where all phenomena (all properties of concepts) are either true or false, and a consistent domain theory by which any proposed fact may be proved to be true or false. The advantage of inductive and abductive inference methods is that they are not necessarily subject to the limitations of first order logic: The world may be viewed more realistically as an open, dynamically changing environment. Concepts may be described in terms of typical features or default values which may be inherited to more specific concepts, and replaced by local values if these later become available. A disadvantage is that the particular inference methods needed has to be defined by the system developer, leading to a complex and often unclear definition of knowledge semantics. (The issue of deductive vs. non-deductive inferences and reasoning is given a deep and thorough discussion by the collection of papers in [18]).

Based upon the three *inference types* described, a range of *inference methods* have been developed in AI. An inference method specifies a way to derive new information from existing knowledge in an actual representation system. Examples are specific methods for chaining of rules, for inheritance of properties in network hierarchies, for matching of concept schemas, etc. Inference methods may be put together to form *inference structures* (like Clancey's inference structure for heuristic classification [10]).

### *2.2. A model of knowledge*

#### *2.2.1. Knowledge types*

A thorough knowledge model of a domain contains different types of knowledge. In order to reason with various knowledge types, the system needs to know their different properties. That is, the system should contain an explicit model (meta model) of its own knowledge structure.

A given piece of knowledge may be classified by its position on a set of knowledge dimensions (figure 3). The following four knowledge dimensions reflects important semantic aspects of knowledge, from the perspective of knowledge-intensive problem solving and learning:

Knowledge *level* refers to whether the knowledge describes concepts of the application domain - called object knowledge - or the use and control of the object knowledge - called control knowledge. Knowledge *depth* refers to a scale from fundamental, theoretical, deep model or 'first principles' knowledge, up to

readily applicable rules of thumb or concrete past experiences that directly associate a solution with a set of problem descriptors. Knowledge *generality* classifies knowledge according to how large a space of the real world it describes. Conceptual model knowledge is typically general knowledge, since it contain class definitions, described by general relationships with other classes. A more shallow type of general knowledge is typically held within heuristic rules. A more specific type of knowledge is the knowledge contained in past episodes and concrete experiences, for example previously solved cases. Knowledge *role* is a dimension related to the use of knowledge. It splits into two subclasses: Descriptive knowledge - defining conceptual structures, stating established facts and assumptions - and operational knowledge, like a direct association between an input descriptor and a goal concept, or an action sequence that may be activated in order to reach a particular goal. The fifth dimension shown in figure 3 is related to the syntactic *form* of the knowledge, i.e. whether it is expressed as a concept network, if-then rules or units of cases.

Explicating these dimensions makes it possible to attack some of the major problems of today's knowledge based systems, by:

- Explicit modelling of problem solving and learning strategies, which enables a system to reason about strategies as well as object knowledge
- Combining shallow and deep knowledge, which may be present both at the control level and at the object level (although control level knowledge typically is shallow)
- Problem solving by combining previous experiences with more generally applicable knowledge
- Relating knowledge to its role in problem solving and learning, extending the notion of knowledge to include methods and procedures as well as concept schemas and heuristic rules

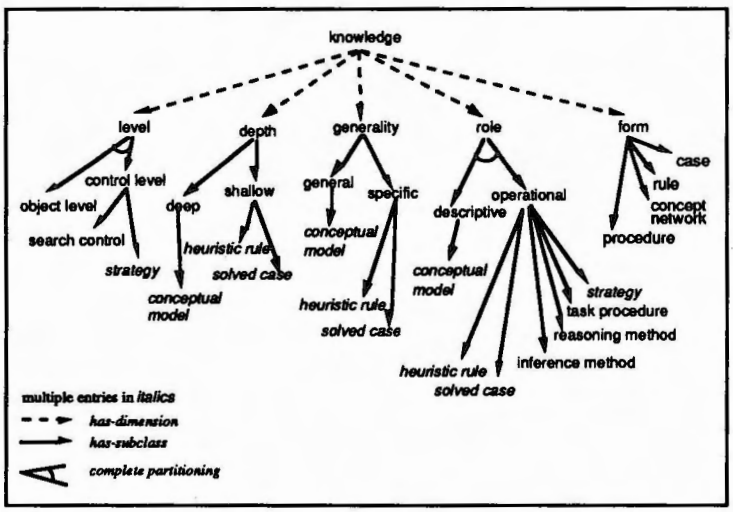



**Figure 3: Dimensions of Knowledge**
The four semantic - and one syntactic (form) - dimensions for characterizing knowledge, described by specializing them into subclasses.

### *2.2.1. The knowledge network*

The term *knowledge model* refers to the total network of concepts and relations of different types - including heuristic rules - as well as the collection of past cases. The nodes in the network are concepts (entities as well as relation type concepts), and the links are relations. A relation is also a concept with its own definition. Cases and rules are coupled to the conceptual network model by having all their descriptors explicitly defined in the model.

A knowledge model should be viewed as a single, tightly coupled network of concept definitions, heuristic rules, past cases, procedures, methods, and strategies. The basis for the system's understanding is the fundament of descriptive knowledge formed by explicit definitions of all terms used in the various submodels.

### 2.2.2. *A model of expertise*

When the 'traditional' notion of knowledge is extended to incorporate reasoning methods and problem solving strategies as well, it is often referred to as *expertise* (e.g. [40]). Human learning is to a large extent an automatic mechanism, but experts also acquire *knowledge about how to learn*, which may also be used deliberately to improve learning. This knowledge is used together with domain knowledge to decide, for example, which parts of an experience are worth remembering, and whether some feasible generalisations should be made.

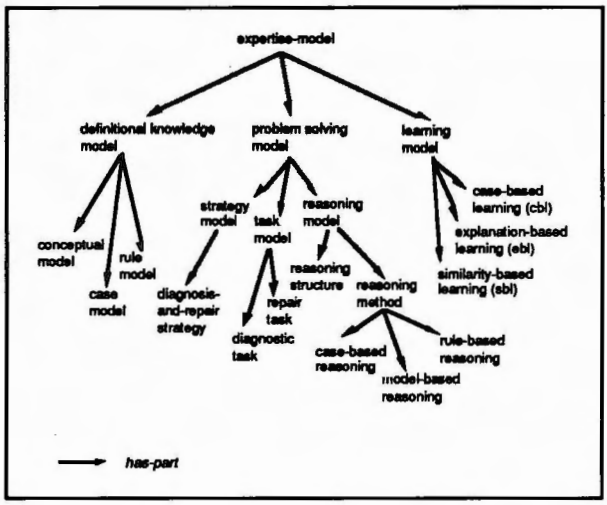



**Figure 4: A Component Model of Expertise**
The figure shows components of the expertise model, with control knowledge for diagnosis-and-repair type of problems.

The extension of the knowledge concept towards expertise, in the sense sketched here, is a consequence of recognizing problem solving and learning methods as knowledge to be modelled and represented, just as factual, object level domain knowledge. In addition to the structural component model of expertise shown in figure 4, a functional model - a methodology for integrating and controlling the submodels - is needed. In the KADS project [6, 40] an approach to modelling of problem solving expertise has been developed, based on four layers of expert knowledge: object domain layer, inference layer, task layer, and strategic layer.

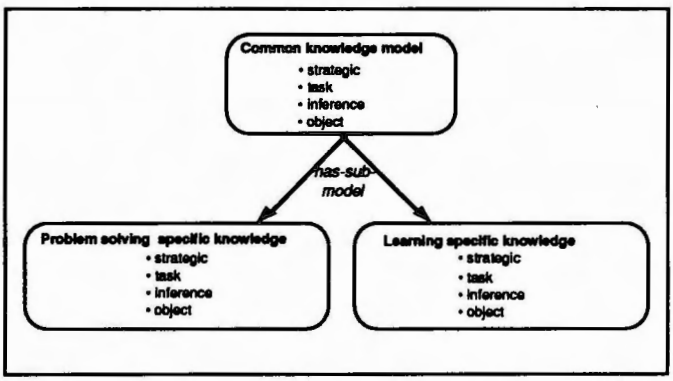



**Figure 5: Integrated Four-Layered Expertise Model**
The KADS four-layer model may be used as a basis for modelling learning competence as well as problem solving competence by integrating separate problem solving and learning knowledge into a common knowledge model.

The four-layer model in KADS is developed as a model of problem solving expertise. Hence, it differs from our expertise model in that learning competence is not incorporated. This is the traditional view of expert knowledge, but viewing learning competence as a kind of knowledge seems the right approach when integrated methods for problem solving and learning are aimed at. However, the generic nature of the four-layer model makes it suitable to describe learning competence as well - as illustrated in figure 5.

The object layer (called 'domain layer' in KADS) basically consists of knowledge in the form of conceptual models, past cases and heuristic rules. Task independent knowledge is a basis for problem solving as well as learning. At the inference level, most methods will be general - applicable to problem solving and learning alike, some will be specific for certain kinds of problem solving and learning methods. Problem solving tasks and learning tasks are different, and will have different task structures. The same is true for strategies, but a top level strategy is also needed, describing how to combine problem solving and learning - e.g. when to solve the problem (a subproblem), and when to (partially) learn. By sharing some knowledge structures between the learning and the problem solving model, both learning and problem solving competence can be described by the four-layer model.

### *2.2.3. Knowledge representation*

Within this framework, representational explicitness is also facilitated by representing relations as a type of concept.

A concept is either an *entity* or a *relation*. A relation is the link in a relationship of two or more concepts. An *attribute-relation* is commonly regarded as a unary relation, describing an intrinsic property of a concept. An attribute relation has no explicit semantical content. For example,

my-car: red, big,old

expresses that my-car has the three attributes red, big and old[1]. This way to express concept properties, features, is common in classical, syntax oriented machine learning methods. Attribute expressions involve implicit relations. A relation like *has-attribute*, *has-feature*, or *has-property* may be assumed as a hidden, unnamed attribute-relation.

Knowledge-intensive problem solving and learning methods should make use of explicit relationships in their understanding and reasoning processes. Attributes should be replaced by semantic relations whenever such relations are known:

my-car: *has-color* red, *has-size* big, *has-age* old.

*Has-color*, etc. are concepts with their own definition within the knowledge model, describing, e.g., what it means to have color. Being able to represent relations as concepts is a requirement for thorough knowledge modelling, and an important characteristic of our framework.

Entities are split into primitive and complex entities. Primitive entities corresponds to semantical primitives of the domain. For example, a *physical object* is an object that occupies a part of physical space, characterized by a particular volume, shape, mass, etc. An *abstract object* is a non-physical object, like a mathematical point or line, a measurement, or a hypothesis. A *state* is a coherent characterization of a part of the world with particular properties, an *event* is a transition between states, and a *process* is a sequence of events. Complex entities range from simple statements (relationships) to large conceptual structures (models).

Relations are also grouped into subclasses with characteristic properties. Properties of relations are important for generating and evaluating explanations (section 2.3.1).

The core frame representation system of CREEK has been implemented on a TI Explorer in CommonLisp. A separate, internal knowledge-base holds an explicit model of generic representational structures like case-frame, rule-frame, transitive-slot, constraint-facet, etc. This knowledge base defines general properties of representational terms, which are used by the system in its control procedures. CREEK is linked to the METAKREK knowledge acquisition environment [34, 1, 39, 25], which enables graphical editing of knowledge structures[2].

---

[1]To be correct, the *implicit attribute names* are colour, size and age, with the *explicit attribute values* as shown.

[2]Another approach to machine learning within METAKREK, also with relevance to the ACKnowledge project [17a] is discussed in [5a].

### *2.3. Problem solving*

This section presents a broad, general model of problem solving, reasoning and learning where the generation and evaluation of explanations play a core part. The approach described partly draws upon results from theories and research on human cognitive processes. This is founded on the belief that a *computational model* for robust problem solving in real world complex domains should to some extent take advantage of existing theories of human problem solving. After all, people have shown to be the most successful problem solvers to date. This is not taking the stance that a goal of AI is to emulate human reasoning per se, the rationale is a pragmatic one: In order to build computer-based problem solvers at an expert level, look for justified and plausible models of human problem solving to start out from.

#### *2.3.1. Explanation*

Classifying an example as an instance of a concept, selecting relevant symptoms from a set of observations, choosing the best diagnostic hypothesis on the basis of relevant findings, assessing the plausibility of a derived result, etc., may be viewed as processes of producing the necessary explanations to derive, support or justify a result. When receiving a patient with the problem "I have this pain in the lower right region of my abdomen, what is the diagnosis?", a physician will typically use his knowledge to explain to himself that it is likely to be an appendicitis, or a strained muscle, or a sub-dermal inflammation, or whatever fault he regards as reasonable given the observations and the current situation context. Generating and evaluating explanations then become an essential part of the reasoning process itself. The central role of explanations in learning and AI in general is discussed by Kodratoff in [13]. Schank (in [31]) argues that the capability of an agent to explain to itself why something is the case, or how a result was derived, is a requirement for understanding, and hence for intelligent behaviour. He further argues that the most important driving force behind learning is an *expectation failure*, where the recognition of a failure is followed by an attempt to explain why the expected result was wrong.

What, then, characterizes a good explanation? There seem to be consensus among various schools that correctness - or truth in a deductive sense - is not a *sufficient* criteria for a good explanation (see the discussion following McDermott's critique of logisism [18]). Is correctness, then, a *necessary* criteria for a good explanation? McDermott argues that an explanation may be good merely by making the observed facts probable, not necessarily proving their truth. Another argument is related to the role of pragmatics: An explanation may be a good one (in a certain context) even if it hides facts that under other circumstances would be regarded as important for understanding a phenomenon. An example is the deliberate hiding or twisting of truth for pedagogical purposes. In the history of science there are a lot of examples of explanations that served perfectly as explanations of phenomena and to the advancement of science, although being false: The earth as the centre of the solar system, an 'ether' substance for propagation of electromagnetic waves, etc. Achinstein (in [Achinstein-85]) argues that:

> "The goodness or worth of an explanation is multidimensional; correctness is only one dimension in an evaluation. An explanation is evaluated by considering whether, or to what extent, certain ends are served. The ends may be quite varied. They may concern what are regarded as universal ideals to be achieved, particularly in science, e.g. truth, simplicity, unification, precision. Other ends are more "pragmatic" ".

This view is supported by Thagaard [36]:

> "Of course, formalization of some sort will be necessary for any computational implementation of scientific knowledge, but it will have to be directed toward procedural issues rather than logical rigour. The emphasis on syntax that is endemic to the view of theories as axiom systems leads to the neglect of semantic considerations crucial to the understanding of conceptual development, and to the neglect of pragmatic considerations that are crucial to justification and explanation."

The notion of *knowledge coherence* - as a relaxation of the formal notion of consistency - has been adopted by several authors (e.g. [10, 16]), but there exists no known models or theories of knowledge coherence applicable to our framework. Thagaard, however, (in [37]) proposes an interesting theory of knowledge coherence called *explanatory coherence*. Coherence, in this theory, is primarily related to a *set* of propositions. It only makes sense to talk about coherence of a single proposition if viewed with respect to another proposition or set of propositions. The notion of *acceptability* is introduced to characterize the coherence of a single proposition with respect to a set of different propositions.

What we need in our modelling framework is a computational model of explanation. This model should be based on relevant general theories of what explanations are, and how they function in problem

solving. Essential parts of such a theoretical basis was described in the preceding section. The model of explanation within our framework is based upon the following fundamental principles:

- Correctness - in the broad sense that includes plausibility - is regarded as a necessary requirement for a good and useful explanation.
- A plausible result is viewed as a product of an abductive reasoning chain, i.e. as a product of an *inference to the best explanation.* This view to abductive reasoning and explanation is close to the views expressed by several philosophers with a computational view to epistemological problems (as briefly reviewed in [37]).
- A body of knowledge need to be *coherent.* in order to be useful for inferencing.An explanation produced within a knowledge model is valid only if it does not 'incohere' with other explanations that explain the same phenomena. The notion of coherence in our model has the same role as *consistency* in a model which assumes complete domain theories. Coherence is a much weaker term than consistence, and expresses a context dependent, global evaluation of a knowledge model. The knowledge model will typically contain submodels that are *locally consistent.*

An explanation in CREEK is a chain of relationships between two concepts, for example:

low-battery-voltage *is-negation-of* high-battery-voltage
engine-does-not-turn *is-negation-of* engine-turns
engine-does-not-start *is-negation-of* engine-starts

high-battery-voltage *causes* starter-motor-turns
starter-motor-turns *causes* engine-turns
engine-turns *is-necessary-requirement-for* engine-starts

explaining why a low battery voltage causes an engine not to start[1].

### *2.3.2. Phases in Problem Solving*

A lot of research in cognitive psychology and artificial intelligence have addressed the issue of breaking the problem solving process into subphases. A basic view in AI is to view problem solving as search in a problem space. The problem space consists of an ultimate goal state, an initial state, and intermediate states representing subgoals and states derived from the initial state. Problem solving, then, involves finding a sequence of operators that transform the initial state into the goal state, thereby achieving the goal. Many problem solving models have been developed to describe how goals may be split into subgoals, how operators are formed or selected, etc. (e.g. the Soar model [15]). A phase model of problem solving - as described here - is a more abstract model of the problem solving process at a strategic level, specifying when to do what, rather than how to do it.

Below, the framework's top level problem solving model is outlined. It is partially based on psychological models, like Polya's model [26]. The model identifies three phases that describe the problem solving process in general, irrespective of whether a problem is solved by direct association, by a more elaborative inferencing process, or both.

A problem solving process has - at the top level - the following three phases:

**A. Understanding the problem**
**B. Generating plausible solutions**
**C. Selecting a good solution**

Although the process starts with A and ends with C, problem solving is generally not a strictly sequential process. It is rather a cyclic activity of A-B-C-A, A-B-A, and B-C-B cycles, which may be recursively called on subproblems.

*A. Understanding the problem*

The first step is an attempt to understand the problem. An agent - human or artifact - competent of solving problems in a domain, is viewed as having a knowledge model - a cognitive model - within which understanding, learning and problem solving takes place (three interesting cognitive models are described in

[1]To make a *useful explanation* out of such a sequence of relationships, an appropriate interpreter is needed in order to draw the necessary intermediate inferences. In this example, the following three *plausible inference rules* are needed:

If A *causes* B, and If B *causes* C, Then A *causes* C
If A *is-necessary-requirement-for* B, Then (negation-of A) *implies* (negation-of B)
If A *causes* B, and If A' *is-negation-of* A, Then A' *implies* (negation-of B).

[29, 12, 21]). The cognitive model holds the structures, concepts and methods of the agent's expertise. In order to understand a problem, the input description has to be interpreted by this model. The interpretation process may be viewed as a series of coherent explanation processes that select relevant descriptors from noisy ones, identify inter-relationships among them, and test if the given and inferred facts make sense.

Understanding then becomes a process of elaborating upon the problem description using the cognitive model: When an agent is presented with a new piece of information, an attempt is made to integrate the new fact into the existing knowledge. This may be viewed as explaining the new fact within the existing cognitive model, by generating confirmations or plausible justifications to support - or reject - the new information. The integration process checks for immediate contradictions and infers implications (associated facts) recursively.

*B. Generating plausible solutions*

The next major phase is to generate a set of plausible solutions, i.e. a set of possible solutions that has been justified in the sense that they achieve the goal without contradicting important constraints. A plausible solution may also have a trace attached to it, that explains (supports) the solution.

Having understood the problem implies that some knowledge structures have been *activated*. To understand a car-starting problem, for example, concepts like car-starting, starter, starting-fault, battery, etc. are likely to have been activated, along with a lot of associated entities, relations, possible faults, repair actions, alternative ways to start a car, etc. Given the observations, the goal of getting the car started, and knowledge about car-starting problems, a competent problem solver will arrive at some candidate solutions to the problem. This may be done by adapting a solution from a previous case, by rule chaining, or by inferencing in a deeper model of knowledge. Further elaboration may take place immediately to strengthen or weaken some of the candidates, for example by creating multiple explanations for supporting an hypothesis or looking for contraindications. The eventual result is a set of hypotheses that are believed strongly enough to be subjects for further evaluation.

*C. Selecting a good solution*

The last stage is an evaluation of plausible solutions in order to select a good solution (preferably the best) to the problem. This phase is characterized by the fact that a single solution candidate has to be chosen (only one solution can be the first one to try out), and a strategy and criteria for selection must be a part of the domain model. The user may play a central role in this phase.

If the selected solution is found unacceptable, a re-iteration of the problem solving process takes place (in principle), where the problem description may be stripped by removing irrelevant descriptors, and expanded by parts of the current system state, like derived consequences of the initial description, unacceptable solution types, and reasonable assumptions.

The figure below summarizes the roles of the three phases:

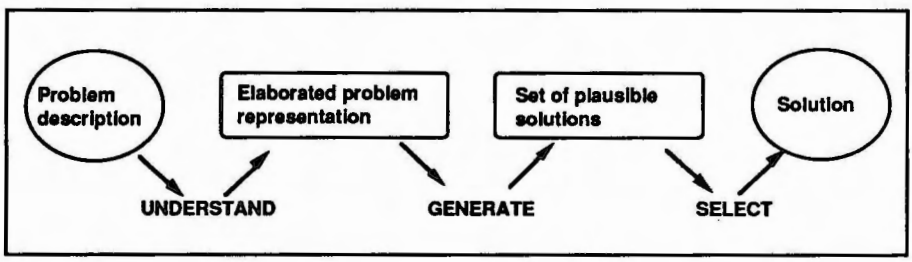



**Figure 6: Main Phases of Problem Solving**
The figure illustrates the UNDERSTAND - GENERATE - SELECT process of knowledge-intensive problem solving, transforming a problem description into a solution.

*2.4. An Explanation Based Model of Reasoning*

A reasoning model is a further specification of some parts of the problem solving process. Within the perspective of the problem solving model just described, a reasoning process is a successive combination of various inference methods (like matching, property inheritance, constraint propagation, rule chaining), guided and supported by explanation methods that focus on the current goal and context of the process. The

reasoning process is based on a combination of the three *reasoning types* model-based reasoning, case-based reasoning and rule-based reasoning.

Clancey's *heuristic classification* model is a well known example of a high level inference structure, what here will be referred to as a *reasoning model*. The basic inference step is the heuristic match, characterized by an experienced relationship or a poorly understood correlation.

Clancey's model is inadequate as a basic reasoning model within our framework, for the following two reasons:

First, the heuristic classification model is motivated by analysing and describing rule-based systems (the model is based on an abstraction of MYCIN [7]). A heuristic rule is a direct association between *input data generalisations* and a *generalised conclusion*, e.g.:

```
(ignition-key turned) AND (lamps (all dark)) ==> electrical-system-fault
```

Our reasoning model needs to cover for case-based reasoning and deeper, model-based reasoning as well. Case-based reasoning involves an associational match between input and conclusion - as rule-based reasoning does - but at the *instance level* rather than at an abstract level:

```
car-starting-case-351
        time  early-morning
        ignition-key  turned
        lamps  (all dark)
        radio  (not working)
        battery-voltage ok
        fault  starter-solenoid-defect
```

Second, even if a heuristic relation expresses an association that has shown to be useful, it may be explainable by a deeper knowledge model, if such a model exists in the system. The kind of knowledge rich systems aimed at in this paper will always have a deeper conceptual model of knowledge, which may be used for justification of heuristic relations as well as deep model reasoning. The interaction between general heuristics and deeper knowledge must therefore be expressible within our reasoning model.Clancey's model does not capture the kind of reasoning and inference methods required to utilize a thorough domain model in this way.

What is needed, then, is a more general reasoning model, where Clancey's model may be viewed as a submodel for one type of reasoning. Other models, like Chandrasekaran's framework based on generic tasks [8], and J. McDermott's model [19] based on a kind of generic task called *role limiting methods*, provides a more general and flexible model of problem solving and reasoning.

The notion of a generic task represents an interesting view to reasoning and problem solving. It differs from some other modular frameworks (e.g. KADS [6]) in that domain knowledge is partitioned into task specific units, coupled to its use and associated with corresponding task specific control strategies and inference methods. The framework described in this paper is based on the assumption that task-specific knowledge is useful and relevant for modelling specific parts of a knowledge model, but also that large portions of knowledge and inferencing is of a general nature. Concrete experience from past cases, for example, is a typical example of task-specific knowledge. However, in order to achieve the high degree of understanding that is required by competent and robust knowledge based systems, a lot of knowledge will have to be of general nature. The role of the more general knowledge is (at the object level) to generate supporting explanations from the general knowledge model when this is needed, and (at the control level) to be the 'glue' necessary for the system to understand the inter-relationships between more specific task-related knowledge (e.g. by explaining why a certain strategy is better than another in a certain situation). In addition, an important role of general knowledge is to be the knowledge fundament to fall back on when more specific knowledge fails or does not apply.

Our model of reasoning emphasizes the generation and evaluation of explanations to support both abstract and concrete associations and elaborate deep model reasoning. Given some findings and a goal, a reasoning process may, at a general level, be described by three sub-processes:

1. **Activating knowledge structures**
2. **Explaining candidate facts**
3. **Focusing on a conclusion**

A reasoning process is viewed as a process of *activating* a certain part of the existing knowledge - including triggering of hypotheses and goals[1], *explaining* to what extent the activated parts form a coherent knowledge-structure, and *focusing* within the explained structure, returning an explicit answer or just the final state of the system, as illustrated in figure 7. Note that this reasoning model is at level below the problem solving model shown in figure 6. Thus, the reasoning within a problem solving step (Generate, say) may involve all three steps of the reasoning model.

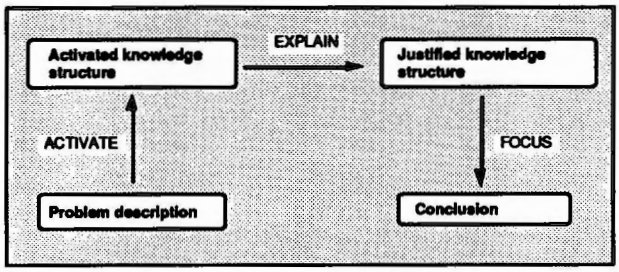



**Figure 7: The Explanation-based Reasoning Model**
The framework's generic model of reasoning, subsuming deep model reasoning as well as model-supported case-based and rule-based reasoning.

Characteristic roles of each phase:

*1. Activating knowledge structures*
ACTIVATE initially sets up the knowledge structure by marking network concepts that matches terms of the input description (data describing the problem, and goals)[1] as active. Other concepts are activated by mechanisms that trigger activation of associated concepts. This spreading of activation has to be controlled in some way, either implicitly by limiting the types and numbers of associative links, or by explicitly controlling the selection of links to spread along dependent on the problem solving state of the system. For example, if a diagnosis system is trying to infer what new measurements to take, it may choose another subset of triggering associations than if it is trying to generate possible diagnostic hypotheses. Explicitly defined concepts as well as relationships (facts) may be activated. Examples of activation spreading is similarity matching of cases, rule chaining, propagation of constraints and deriving of consequences in semantic networks. The purpose of this step is to activate possible candidate concepts. Even if some control knowledge is available to constrain and guide the process, the activation phase is not necessarily a very knowledge-intensive process.

*2. Explaining candidate facts*
EXPLAIN starts to work on the activated concepts, and its job is to use the knowledge (and the user, if necessary) to justify, confirm or reject candidate facts by producing supporting explanations. For example, if two activated facts contradict, the one with the strongest explanation will be chosen. The strength of explanatory support depends on the strength of single explanation chains as well as the number of alternative explanations that support the fact. The goal of the reasoning process, e.g. the particular subproblem to solve, focuses the generation of explanations. The outcome of the explanation phase is a coherent knowledge structure and a set of supported hypotheses that are good candidates for achieving the goal of the reasoning process.

*3. Focusing on a conclusion*
FOCUS is the final step, that use constraints and pragmatic criteria on the conclusion to check a candidate conclusion, or select a conclusion if more than one candidate for satisfying the goal came out of the explaining phase. While EXPLAIN generated support and evaluated its hypotheses according to whether a

[1]Goals are end-states to be achieved, while hypotheses are assertions to be confirmed or rejected. The notions of goal and hypothesis are to be interpreted in a general sense. They cover intermediate goals and hypotheses set up by the system during reasoning, as well as top level goals (like 'find-fault)'. A hypothesis may be an expected value of a problem descriptor as well as a possible solution to the problem.

[1]In addition, some problem solvers accept an input type referred to as *support knowledge*, i.e. input intended as support for the problem solving, reasoning and/or learning process. Support knowledge is control level knowledge like what problem solving strategy to use (first), what task model or reasoning method to apply, etc. Since the role of support knowledge is to take some of the burden away from the system's own control level reasoning, and since our framework does not assume that support knowledge is entered, it is not further discussed in this paper.

conclusion make sense and may be useful, FOCUS refines and adapts the candidate set in order to pick the best (i.e. most plausible) conclusion.

This three-phase model focuses on knowledge-intensive reasoning, but is also general enough to subsume most reasoning schemes used in AI systems. ACTIVATE may trigger hypotheses in a data-driven or goal-driven manner, by chaining of rules, retrieving of past cases, or inheritance in hierarchical knowledge models. It may associate facts on the basis of shallow or deep knowledge, and at the object level or control level. Many expert systems (e.g. OPS5-based, PROSPECTOR-like and MYCIN-like systems) do most of their reasoning within ACTIVATE. The EXPLAIN step play a minor role in these systems, since they generally do associational reasoning based on shallow, compiled knowledge. SELECT may be a calculation of probabilities or a decision based on control heuristics.

The model should not be viewed as a strictly sequential process. For example, instead of initially activating all possibly relevant structures, a reasoning process may activate one or a few structures, try to explain the consequences and expectations they generate, activate new structures, etc.

### *2.4.2. Reasoning from Past Cases*

The reasoning model described above subsumes knowledge-intensive case-based reasoning. A generic model of reasoning from past cases contains the following three steps:

1. Activate: *Retrieve* a set of cases that matches the current problem according to some criteria. This step activates cases that will be subject for further examination.
2. Explain: *Evaluate* the applicability (relevance) of retrieved cases to the current problem. This involves looking for contradictions, checking constraints, generating and testing expectations set up by the retrieved case.
3. Focus: *Adapt* the solution (or solution method) of the case to the new problem. The adaption may be a simple transfer of the past solution to the new problem, or an elaborative process of modifying the solution to better fit the current problem. A modification process would typically involve a generalization of the past solution followed by a specialization satisfying constraints of the current problem.

At a more detailed level, each step in the RETRIEVE-EVALUATE-ADAPT cycle should be regarded as separate tasks. The ACTIVATE-EXPLAIN-FOCUS process - being a general model of explanation-based reasoning - is then recursively applicable to each task[1]. This illustrates the generic role of the reasoning model as a generator and evaluator of explanations.

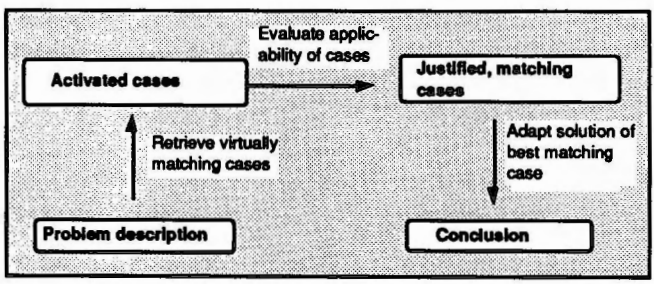



**Figure 8: Main Steps in Reasoning from Past Cases**

## *2.5. Learning*

### *2.5.1. An Explanation Based Model of Sustained Learning*

The learning model is tightly integrated with problem solving. The focus is on how to capture relevant experience from a problem solving session, and make it available for solving similar problems later. The

[1]The whole process is controlled at the task level, using criteria for evaluating the conclusions of each sub-process - along with threshold values for their acceptance/rejection.

learning model integrates a case-based and explanation-based learning approach, with a learning apprentice view.

Schank has studied explanation mechanisms related to learning in people [30, 32], and views learning as a process of explaining a failed expectation[1]:

"What situation needs to be explained? People have powerful models of the world. Through these models, which are based on the accumulated set of experiences that a person has had, new experiences are interpreted. When the new experiences that a person perceives fit nicely into the framework of expectations that has been derived from experience, an understander has little problem understanding. However, often a new experience is anomalous in some way; it doesn't correspond to what we expect. In that case, we must reevaluate what is going on. We must attempt to explain why we were wring in our expectations. We must do this, or we will fail to grow as a result of experiences. Learning requires expectation failure and the explanation of expectation failure."

Within our framework, learning in a knowledge rich domain is viewed as a process of

1. Observing phenomena
2. Inferring consequences, i.e. generating expectations
3. Trying to explain what is observed and inferred
4. When expectations fail, trying alternative explanations or asking for more information from the external environment
5. Extracting new knowledge relevant for future problem solving
6. Updating the knowledge model

When the learning 'starts', the system state is an activated structure of knowledge representing an understanding of the actual problem, and a plausible, explained solution. This activated structure should not be regarded as fixed, however, since the learning task may infer new information that lead to contradictions within the structure. Resolving such contradictions may introduce changes in the knowledge structure. Given this assumption, the learning process may be described generally as follows:

**1. Extracting learning sources**
**2. Constructing new knowledge structures**
**3. Storing and indexing**

These steps - with corresponding types of input/output data - are illustrated in figure 9.

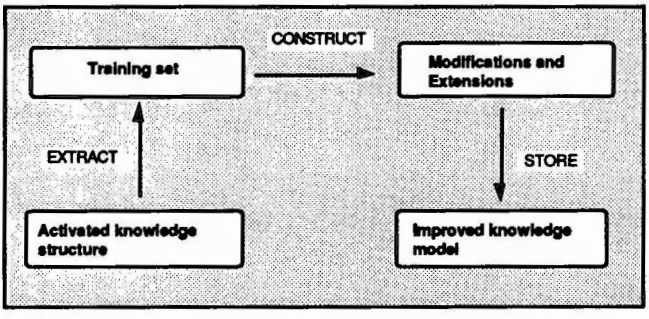



**Figure 9: A Model of Learning from Experience**
The knowledge structure that has been activated during problem solving is the primary source of learning. A subset of structures (e.g. input descriptors, goals, successful solution path, failed attempts) is extracted as the 'training set' to actually learn from. The middle step - CONSTRUCT - may be viewed as the primary learning step. The new case and/or general knowledge structures that has been constructed are finally associated with index concepts (important features, remindings to similar cases, etc.) for later retrieval.

*1. Extracting learning sources*

[1]From [Scank-86c], page 144.

This subprocess is active throughout problem solving, and its task is to keep track of information and knowledge that later will be used as sources of learning. Some methods use the problem description given as input to the system together with the problem solution as the only type of candidate examples for learning, while other methods also learn from inferred facts and traces of the problem solving process. The set of learning sources form a selected training set that is given as input to the constructive learning method.

This step is absent in knowledge-poor, similarity-based learning methods: Since no knowledge model is available to guide the selection and extraction of relevant training instances, they are all regarded as equally relevant.

*2. Constructing new knowledge structures*

A method for storing a problem solving experience in a new case may be characterized by the answers to the following four questions:

- What determines whether a new case should be constructed?
  - degree of dissimilarity with closest case
  - failed attempt to solve problem
  - successful attempt to solve problem
- What is to be stored in a case?
  - problem description
  - solution
  - trace of problem solving path
  - dead ends (attempted paths that failed)
  - explanation chains
  - dependencies, preconditions for activation
  - remindings to other cases
- How general is the contents of a case?
  - cases are distinct individuals
  - cases are hierarchical structures
  - cases are generalised to rules
- How to modify the existing case structure?
  - changing strength of remindings
  - merging with new case

Within the framework presented here, explanation-based methods are related to the notion of *knowledge integration*, i.e. integrating a new piece of knowledge by explaining it using existing knowledge. In other words, new knowledge is learned in the sense that it is 'understood' by being explained on the basis of what is already known. Models of learning as knowledge integration has been developed, e.g., by Murray and Porter [24], and Van de Velde [38]. The model of Murray and Porter is particularly interesting, since its development has been guided by experiments within a very knowledge-intensive environment: The CYC system [17]. Knowledge integration is also an important issue for knowledge acquisition tools in general. In the ACKnowledge project [17a] - an ESPRIT project aimed at developing an advanced knowledge engineering workbench - integrating knowledge acquired by different tools into an existing knowledge base is a crucial problem that needs to be solved. A method, partly based on Murray's model, has been studied [11].

*3. Storing and indexing*

Learning is not successful unless updates and refinements are available for new problems. The storing problem is to decide what form the knowledge should be saved in, and where it should be put. Essential questions for characterizing a case-based learner's storing and indexing scheme are:

- Are cases separate knowledge units, or are they virtual structures of connected nodes?
- How are cases indexed (if at all)?
- What kind of indexing vocabulary is used?

The storing problem related to indexing for efficient later retrieval, is the main task here. If knowledge is learned in a generalised form, and integrated into a conceptual knowledge model or a rule set, the problem is what existing concepts and rule structures to update. If a case is to be stored, the problem is what concepts to use as pointers to the case, and how structure the index network. In both situations it is a question of creating the necessary remindings and associations to new pieces of knowledge (concepts, cases, rules).

### *2.5.2. Learning by Retaining Past Cases*

It should be clear from the previous section that case-based learning is the primary learning paradigm within this framework. In order to cover the integration of explanation-based methods, The EXTRACT-CONSTRUCT-STORE model illustrated in figure 9 subsumes case-based as well as explanation-based

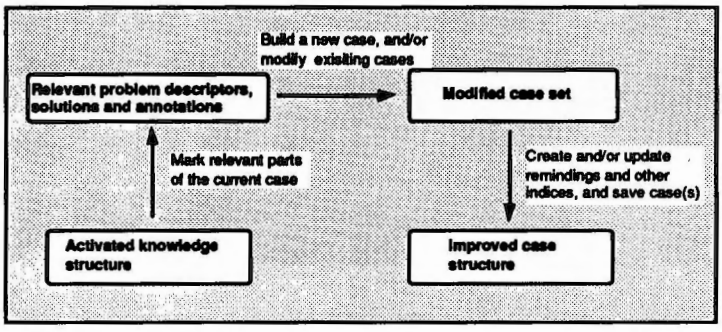



**Figure 10: Main Steps in Learning by Retaining Past Cases**

learning methods. Focusing on the case-learning mechanism alone, the model gets specialized to the one illustrated in figure 10. In CREEK, the cases are stored and indexed by a method based on the PROTOS [5] model. Each case is indexed to by a set of remindings, i.e. features that has occured as input descriptors. Remindnigs may point directly to a problem solution instead of a case.

### *2.6. An integrated model of problem solving, reasoning and learning*

Figure 11 summarizes the framework's model of problem solving, reasoning and learning. It shows how the submodels presented in this chapter interact and cooperate. The middle part of the figure shows the basic problem solving steps: UNDERSTAND-GENERATE-SELECT. The underlying reasoning process within each phase is represented by the ACTIVATE-EXPLAIN-FOCUS cycle. During problem solving, and after the problem has been solved, the learning process extracts relevant sources for learning, which is used to construct new (or modified) knowledge structures. The final step of the EXTRACT-CONSTRUCT-STORE learning process is to index and save new knowledge constructs for retrieval in future problem solving.

Based on the framework presented here, a system architecture (the CREEK system) has been designed [2,3]. The CREEK system addresses classification problems, in particular problems involving diagnosis and repair tasks. A fundamental component of the architecture is a common knowledge 'pool' in which all conceptual knowledge is defined. Four different modules represent, respectively ,an *object level model of the domain* (concepts, cases and rules - where the concepts are part of the common knowledge pool), and three specific control level models: A *diagnosis-and-repair process model*, a *combined reasoning model*, and a *learning model*. The control models contain definitions of control level concepts, rule and procedures. So far, the representation system and a set of basic inference methods for reasoning and learning have been implemented[1].

The diagnosis-and-repair model is a generic knowledge model of diagnosis and repair tasks. It contains concepts like test-result, hypothesis, diagnosis, repair-action, etc., and how such concepts are interrelated. The combined reasoning model contains a strategy for deciding what type of knowledge to apply first, given a particular problem. Basically, case-based reasoning is tried if the problem features give a reminding to a case that is strong enough. Problem features are input features and features derived from input features by spreading activation along a particular set of relation in the conceptual model. The learning model contains concepts and heuristics to guide the extraction of knowledge to store in a case, to control the feature generalization process during merging of cases, etc.

The problem of oil well drilling fluid (mud) supervision, fault finding and treatment has been the application in focus when developing the CREEK system. This is a complex diagnostic task, where the processes of diagnosis and treatment typically are interleaved. The problem is open, has a weak domain theory, and problem solving requires interaction with the user. The application domain is summarized in [1],

[1]As a part of their Master thesis research projects on combined case-based and inductive learning, and on multi-paradigm reasoning, respectively, Velitchko Koulitchev [15a] and Christian Larssen [15b] wrote a case matcher and generalizer, and integrated a rule chainer within the CREEK representation system.

which also contains an example of how to represent object- and control level knowledge within the CREEK representation system.

Current research in knowledge-intensive, combined methods for problem solving and learning that incorporate case-based reasoning include the PROTOS [5], CASEY [14], JULIA [12b], CHEF [11a], and GREBE [5b]. Characteristics of the CREEK approach - compared to these systems - is the strong role of a thorough and comprehensive model of general, conceptual knowledge for support of the case-based (and -to some extent - rule-based) processes. Further, CREEK has an explicit model of problem solving strategy (not elaborated in this paper). The learning method in CREEK basically learns new cases, but is also able to update its general domain model through interaction with the user. None of the other approaches listed above unifies these properties into a single computational model.

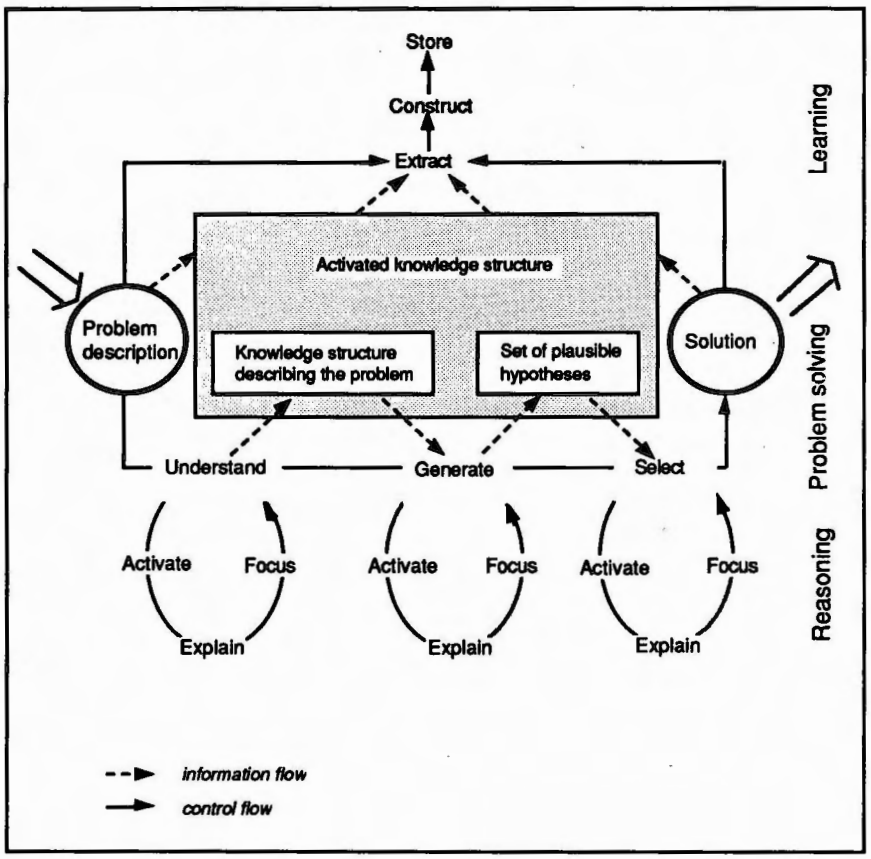



**Figure 11: The Framework's Model of Integrated Learning, Problem Solving and Reasoning**

The figure illustrates the top level flow of control and information in problem solving and learning. For clarity, only some of the interactions between processes and the activated parts of the knowledge base is shown: the information flow to and from problem solving processes, and input to the initial learning process. The activated knowledge structure interacts with all subprocesses, and the gray area in the figure may be pictured as lying beneath all subprocesses shown.

## 3. Conclusion

The framework and computational model presented in this paper forms a basis for developing knowledge-intensive problem solvers that learn from experience, and that are able to use previously solved problem cases to solve new problems. We have argued that such a conceptualization of a knowledge-based

system is a significant step in development of more competent, robust, and adaptive systems. That is, the kind of systems we will need to meet the challenge of expert systems for real world, complex problems.

## 4. References


[1] Geir Aakvik, Mette Vestli, Inge Nordbø, Ingeborg Sølvberg, Tore Amble, Agnar Aamodt: METAKREK; METATOOL documentation. ELAB-RUNIT Report STF14 A88055. SINTEF, Trondheim, 1988.

[2] Agnar Aamodt: Towards robust expert systems that learn from experience - an architectural framework. In John Boose, Brian Gaines, Jean-Gabriel Ganascia (eds.): *EKAW-89; Third European Knowledge Aquisition for Knowledge-Based Systems Workshop*, Paris, July 1989. pp 311-326.

[3] Agnar Aamodt: Knowledge-Intensive Case-Based Reasoning and Sustained Learning. To appear in: *Proceedings of the 9th European Conference on Artificial Intelligence, ECAI-90*. Stockholm, August, 1990.

[4] Ray Bareiss, Bruce Porter, Creg Wier: PROTOS - an exemplar based learning apprentice. In *Proceedings 2nd Knowledge acquisition for knowledge based systems workshop*. Banff, Canada, 1987. Also in: *International Journal of Man-Machine Studies*, Vol. 29, 1988, pp. 549-561.

[5] 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.

[5a] Gunnar Blix, Ingeborg Sølvberg: Learning in a knowledge acquisition toolkit. *Proceedings of the SCAI-89, Scandinavian Conference on Artificial Intelligence - 89, Tampere, Finland, June, 1989. pp 163-176.*

[5b] Karl Branting: *The role of explanations in reasoning from legal precedents.* In: Janet Kolodner (ed.): *Case based reasoning. Proceedings from a workshop*, Clearwater Beach, Florida, May 1988. Morgan Kaufmann Publ. pp. 94-103.

[6] Joust Breuker, Bob Wielinga: Models of expertise in knowledge acquisition. In Guida, G. (ed.): *Topics in Expert System Design.* Elsevier/North-Holland, 1989. pp 265-295.

[7] Bruce Buchanan, Edward Shortliffe (eds.): Rule base expert systems; the MYCIN experiment of the Stanfor Heuristic Programming Project. Addison-Wesley, 1984.

[8] B. Chandrasekaran: Towards a functional architecture for intelligence based on generic information processing tasks. *Proceedings IJCAI-87.* p 1183-1192.

[10] William Clancey: Heuristic classification. *Artificial Intelligence*, Vol.27, no.3, 1985. pp 289-350.

[11] Jorun Eggen, Astrid M. Lundteigen, Mette Mehus: Integration of knowledge from different knowledge acquisition tools. In *EKAW-90, Fourth European Knowledge Aquisition for Knowledge-Based Systems Workshop*, Amsterdam, June, 1990. (This proceedings).

[11a] Kristion J. Hammond: CHEF; a model of case-based planning. *Proceedings of AAAI-86*. Morgan Kaufmann, 1986. pp 267-271.

[12] P. N. Johnson-Laird: Mental models; towards a cognitive science of language, inference, and consiousness. Cambridge University Press, 1983.

[12a] Janet Kolodner: Maintaining organization in a dynamic long-term memory. *Cognitive Science*, Vol.7, s.243-280. 1983.

[12b] 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.

[13] Yves Kodratoff: Is AI a sub-field of computer science - or is AI the science of explanations? In I. Bratko, N. Lavac (eds): *Progress in machine learning; procceedings from the 2nd European working session on learning*, Bled, Yugoslavia, May 1987. Sigma Press, 1987. pp 91-106.

[14] Phyllis Koton: Using experience in learning and problem solving. Massachusets Institute of Technology, Laboratory of Computer Science (Ph.D. diss, October 1988). MIT/LCS/TR-441. 1989.

[15] John Laird, Paul Rosenbloom, Allen Newell: Universal subgoaling and chunking; The automatic generation and learning of goal hierarchies. Kluwer Academic Publ., 1986.

[15a] Velitchko Koulitchev: Generalization in case-based machine learning. RIK 90-3, ELAB-RUNIT Report STF 40A90049. Trondheim, 1990.

[15b] Christian Larssen: Multi-paradigm reasoning. ELAB-RUNIT Report 1990 (Forthcoming).

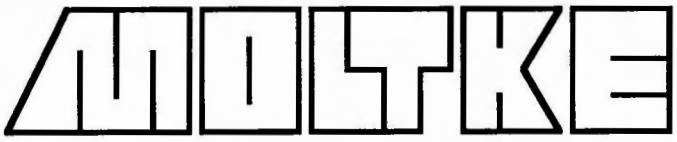
[16] Doug Lenat, Edward Feigenbaum: On the thresholds of knowledge. *Proceeding of IJCAI 1987.* s.1173-1182.

[17] Doug Lenat, M. Prakash, Mary Shepherd: CYC - using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks. *AI Magazine*, Winter 1986. pp. 65-85

[17a] Jean-Charles Marty et. al.: The ACKnowledge Project; improving the knowledge acquisition process. ESPRIT-II Technical Report ACK-CSI-WID-RR-001-B, Cap Sesa Innovation, 1989.

[18] Drew McDermott: A critique of pure reason. *Journal of Computational Intelligence*, Vol3, no.3, Aug. 1987. pp. 151-160. (Target article followed by commentaries from several other AI researchers, and McDermott's final reply. pp 161-237.)
[19] John McDermott: Preliminary steps toward a taxonomy of problem solving methods. In S. Marcus (ed.): *Automating knowledge acquisition for expert systems*. Kluwer Academic Publ., 1988. pp 226-266.
[20] Ryszard Michalski: A theory and methodology of inductive learning. *Machine Learning*, Vol1. pp .83-134. 1983. Morgan Kaufmann.
[21] Marvin Minsky: *The society of mind*. Simon and Schuster, 1988.
[22] Tom Mitchell: Generalization as search. *Artificial Intelligence*, Vol. 18, No.2, March 1988. pp. 203-226.
[23] Katharina Morik: Sloppy Modelling. In: K.Morik (ed): *Knowledge representation and organization in machine learning. Lecture notes in artificial intelligence, 347*. Springer, 1989. pp 107-134.
[24] Kenneth Murray, Bruce Porter: Developing a tool for knowledge integration; initial results. *Proceedings from the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, November 1988. 13 pgs.
[25] Inge Nordbø, Mette Vestli, Ingeborg Sølvberg: METAMETH; Methodology for knowledge engineering. In John Boose, Brian Gaines, Jean-Gabriel Ganascia (eds.): *EKAW-89; Third European Workshop on Knowledge-Based Systems*, Paris, July 1989.
[26] Polya: *How to solve it*. 1957.
[27] 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.
[27a] J. Ross Quinlan: Induction of decision trees. *Machine Learning* (journal), Vol.1 s.81-106. 1986.
[28] Case-based reasoning, from DARPA Machine Learning Program Plan. By: Edwina Rissland, Janet Kolodner, David Waltz. In: *Proceedings from the Case-Based Reasoning Workshop*, Pensacola Beach, Florida, May-June 1989. Sponsored by DARPA. Morgan Kaufmann, 1989. pp 1-13.
[29] Roger Schank: *Dynamic memory*. Cambrigde University Press. 1982.
[30] Roger Schank: *Explanation patterns - understanding mechanically and creatively*. Lawrence Erlbaum, 1986.
[31] 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 contains commentaries by several authors.
[32] Roger Schank: Explanation; a first pass. In J.L. Kolodner, C.K. Riesbeck (eds.): *Experience, memory and reasoning*. Lawrence Erlbaum, 1986. pp. 139-165.
[33] Edward Smith, Douglas Medin: *Categories and concepts*. Harvard University Press. 1981.
[34] 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.
[35] Gheorghe Tecuci: DISCIPLE; a theory, methodology and system for learning expert knowledge. Universite de Paris-Sud, Centre d'Orsay. These presente pour obyenir le titre de Docteur en Science, Juillet 1988.
[36] Paul Thagaard: *Computational Philosophy of Science*: MIT Press/Bradford Books. 1988.
[37] Paul Thagaard: Explanatory Coherence. CSL Report 16, Revised February 1989. Princeton University, Cognitive Science Laboratory. (To appear in Behavioral and Brain Sciences). 1989.
[38] Walter Van de Velde: Learning through progressive refinement. In: *EWSL 88; Proceedings of the Third European Working Session on Learning*, Pitman, 1988. pp 211-226.
[39] Mette Vestli, Inge Romslo, Ingeborg Sølvberg, Magnhild Prytz, Eli Kjøbli, Jorun Eggen, Harald Johnsen, Inge Nordbø, HJens Gleditch: METAKREK - A knowledge acquisition study. *Proceedings Expert systems and their applications, 9th international workshop*. Avignon, France, mai-juin 1989.
[40] Bob Wielinga, Joust Breuker: Models of Expertise. *Proceedings from ECAI-86*, Brighton, England. 1986. pp. 306-318.
[41] Ludvig Wittgenstein: *Philosophical investigations*. Blackwell, 1953. pp. 31-34.

# Multiple Knowledge Acquisition Strategies in



[1]

Klaus-Dieter Althoff, Frank Maurer, Robert Rehbold
University of Kaiserslautern
Dept. of Computer Science
P.O. Box 3049, D-6750 Kaiserslautern
West Germany

**Abstract**

In this paper we will present a design model (in the sense of KADS) for the domain of technical diagnosis. Based on this we will describe the fully implemented expert system shell MOLTKE 3.0, which integrates common knowledge acquisition methods with techniques developed in the fields of *Model-Based Diagnosis* and *Machine Learning*, especially *Case-Based Reasoning*.

## 1. Introduction

When starting a real world expert system project knowledge acquisition is the bottleneck. A main reason for this is the gap between the languages of the domain expert and that used for the implementation by the knowledge engineer. This gap reflects the difference between the cognitive models of the application area the two involved persons have. According to the KADS-group to overcome these shortcomings a conceptual model of the domain has to be developed [1]. Then this conceptual model has to be transfered into a design model.

We think that developing a conceptual model has a few drawbacks:

- The knowledge of an expert includes static and dynamic parts. Static knowledge can be represented in semantic networks whereas it is not easy to express the dynamic parts which describe how a problem is solved. The interpretation model often lacks a clear (i.e. formal) semantics.
- If people express knowledge without having an inference engine in mind the transfer of the developed representation into a running expert system may be hard. It becomes harder the bigger the gap between the conceptual model and the language of the used shell is.

---

[1] The work presented herein was partially supported by the Deutsche Forschungsgemeinschaft, Sonderforschungsbereich 314: "Artificial Intelligence- Knowledge-Based Systems", projects X6 and X9.

- Primitives of particular technical application domains are easily found, e.g. in diagnosis one always talks about symptoms, faults and tests.

Our approach to reduce the knowledge acquisition bottleneck is to describe a design model which reflects the primitives of the application domain. It has to include an inference engine for the knowledge. The task of knowledge acquisition is to fill the templates of the shell with the expert's knowledge. If the primitives really reflect the concepts of the application area the communication between the knowledge engineer and the expert is improved because they speak the same language.

Starting from a real world application we developed a design model for technical diagnosis, implemented its basic vocabulary and an interpreter. The MOLTKE 3.0 Shell, which is described in the next section, reflects the application domain concepts like symptoms, tests and failures. Building up on this kernel more sophisticated tools were developed:

- A compiler which generates the core of an expert system for diagnosis out of a deep model of the technical device.
- Case-based reasoning techniques to enlarge the knowledge acquisition process.

As techniques of three general approaches (deep modelling, machine learning and manual knowledge acquisition) are fully integrated, the MOLTKE system is able to use multiple sources of knowledge.

## 2. A Design Model for Technical Diagnosis

In the MOLTKE project we defined a design model for technical diagnosis. We believe that diagnosis can be described as follows:

**Diagnosis = Classification + Test Selection.**

The state of a technical device is described by a list of *symptoms* (where symptom means something that is measurable, e.g. voltage). A symptom can be low level, e.g. the real voltage at a special measuring point, or more abstract, e.g. "voltage is too high". Therefore a shell for diagnosis has to include mechanisms for data abstraction. Furthermore there is a need for expressing time-dependent symptoms, so called temporally distributed symptoms (TDS) (the integration of TDS in MOLTKE 3.0 is described in [2]).

At any point in time a symptom can have only one *value*. A list of actual symptom values is called a *situation*. In a situation a symptom may also be unknown. A failure in a technical device can be expressed in terms of particular values. Therefore, a *fault hypothesis* is correlated with a set of symptom values. If the actual situation matches these, the failure is established.

Often some symptom values determine others, e.g. if the light in a room is on one can infer that the wires are working. For that reason an expert system must handle relations between symptom values. Moreover this reduces the search space because the determined symptoms need not to be asked.

The search space for diagnosis is described by a graph where nodes are situations and arcs between situations are labelled with a test. A *test* determines the value of symptoms (by asking the user or by getting sensor information). Tests are executed if according to the actual situation no failure can be proved. In technical diagnosis the processing of symptom values must be both demand-driven (the system decides what to measure next) and data-driven (asynchronous incoming data is processed immediately).

Diagnosis is a process: a situation where no failure is proved must be transformed by executing tests into one where the description of a fault matches the actual symptom values. A *strategy* describes the

order of testing and should reflect the experience and heuristics of diagnostic experts as e.g. service technicians.

Modularisation always is a means of software engineering which improves the maintainability of a program. Expert systems shall use the advantages of modularisation. Therefore, knowledge about failures and about strategies shall be separated for reasons of a horizontal modularisation. Vertically the expert system is split in a hierarchy of rough, intermediate, and final diagnoses. The modularisation helps to integrate the knowledge of multiple experts.

## 3. The MOLTKE Project - An Overview

MOLTKE means MQdels, Learning and Temporal Knowledge in Expert systems for technical diagnosis. Within the MOLTKE project (see e.g. [3,4]) we developed an expert system toolbox - the MOLTKE system - and several applications. Figure 1 gives an overview of the MOLTKE system. The shaded part in the center is the kernel, the MOLTKE 3.0 shell. The model component on the left hand side and the case-based reasoning part on the right hand side are described later in this paper.

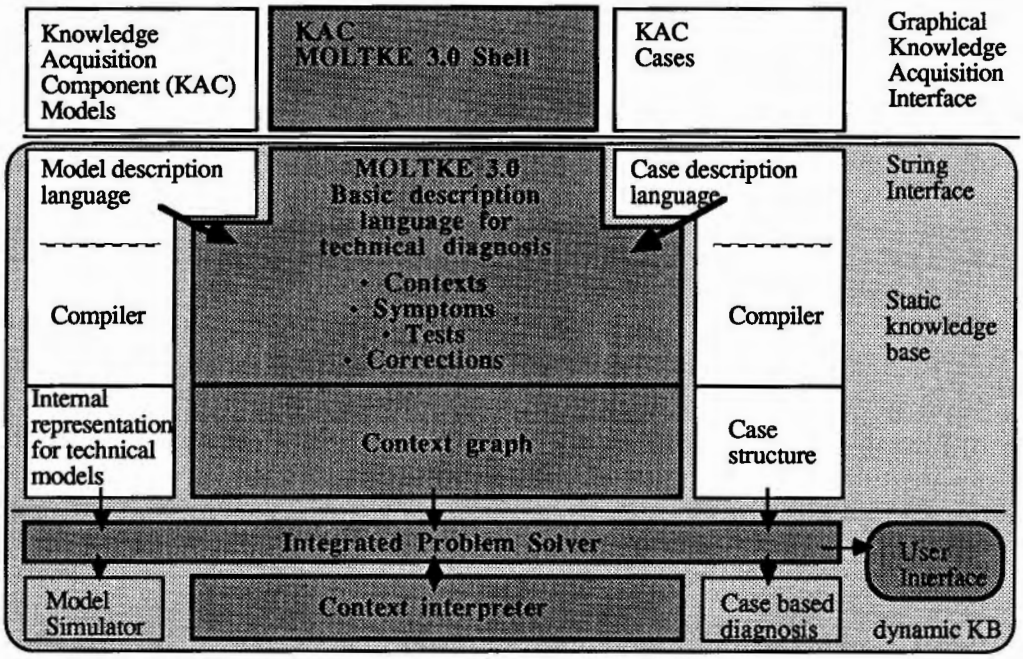



Figure 1: The MOLTKE System

The MOLTKE system is a modularized toolbox for the development of expert systems in the domain of technical diagnosis. It integrates common knowledge acquisition techniques with deep modelling and machine learning. For the interpretation of a knowledge base several interpreters were implemented which allow to integrate different sources of knowledge (see chapter 8).

# 4. The MOLTKE 3.0 Shell

## 4.1. The Static Knowledge Base

The static knowledge base represents the knowledge about the technical system. Here we describe its basic vocabulary:

A *symptom class* relates a name to a list of possible values (e.g.Valve --> {open, closed}) whereas *symptom instances* reflect the actual state of a part of the technical device (e.g. Valve 21Y5 --> open). The actual value may be either *unknown* or an element of the possible value list in the corresponding class.

The set of all symptom instances is called a *situation*. In the context of predicate calculus the actual situation is the base for the interpretation of a *language of formulas*. It stores the bindings of the variables[1]. We use a three-valued logic with *true, false* and *unknown*. For reasons of an efficient evaluation all formulas are stored in network similar to the Rete-Network of OPS5.

A *test* ascertains the value of one or more symptom instances[2]. Every test has an associated precondition (e.g. for specifying that a test can only be executed if the technician has a particular level of skill we would use a precondition like (= skill level high)). A test is only executed if the corresponding precondition is evaluated to true. The sequence of testing is determined by a set of ordering rules where the left hand side is a formula and the right part contains the symptom instance to test (e.g. (if (= Valve 21Y5 open) then OUT30 test)).

To express relations between symptom values *shortcut rules* are used (e.g. (if (= lamp room on) then wires := working)). To express uncertainty every rule may have a list of associated qualitative certainty factors called determination factors[3]. Uncertain shortcut rules are called *partial* in opposite to the *total* ones which are certain with respect to the underlying deep model. These rules are called shortcuts because they abbreviate the diagnostic process, i.e. symptom values which are determined by shortcut rules need not to be asked anymore.

*Contexts* are one of the means for modularisation in MOLTKE 3.0. A context represents a rough, intermediate, or final diagnosis. If its precondition is true, the associated failure is said to be proven and the related *correction* is executed. Any context contains a set of ordering rules which locally prescribe the strategy of testing. Additionally, a context includes a set of shortcut rules. Every context contains a slot for a *context interpreter* which determines the next symptom to be tested. The default context interpreter uses the ordering rules for test selection[4].

A correction describes what has to be done when a special fault occurs. Up to now a description how to repair the device is given to the user. For monitoring tasks this may easily be extended to manipulating control units.

The contexts are organized in a *context graph*. Its arcs have the semantics "is-refinement-of" (e.g. the context *failure-in-electric* is a refinement of *failure-in-car*.).

---

1 Every symptom instance is a variable in the calculus.

2 In the formal sense of modal logic a test describes the transition from one world to another where the interpretation of the formulas differs in the bindings of the examined variables.

3 The processing of these factors is described in section 7.5.

4 We developed neural network associators and model-based components which can replace the default context interpreter.

### 4.2. The Interpreter

The processing of a knowledge base is done by a global interpreter which is easy to adapt and to maintain because it is organized in small modules. The default interpreter realizes an establish-and-refine-strategy (see figure 2).

```
Algorithm of the default interpreter:

actualContext := Root of context graph;
while actualContext not a final diagnosis do
        process all shortcut rules with a true precondition;
        rule := select-an-ordering-rule(actualContext, actualSituation);
        value := test-the-selected-symptom(rule);
        propagate the symptom value through a rete-network;
        if the actualContext has refinements with a true precondition
                then actualContext := this refinement;
        if all refinements of the actual context are false
                then backtracking;
```

Figure 2: The algorithm of the global interpreter

The diagnostic process walks through the context graph by testing symptoms according to the ordering rules of the actual context and switching to a refinement when its precondition becomes (the logical value) true. If a leaf is reached the system stops.

The system is able to reset a symptom value to unknown and to reject all inferences by shortcut rules based on its old value. This facility is used when the user or the system retracts uncertain symptom values.

Sensor data are propagated through the formula network when they arrive[1]. Every time the system asks for a special symptom value the user is able to enter the value of another symptom instance. According to the actual state the system determines the actual context.

## 5. Knowledge Acquisition with the MOLTKE Kernel

The MOLTKE 3.0 kernel includes a comfortable knowledge base browser based on the facilities of the Smalltalk-80 system. The browser combines a menu-oriented interface with different editors for symptoms, contexts, tests, etc. Editing an object means filling a template. In a graphic window the context graph can be edited. The editors include syntax- and type-checkers. If other objects are referenced[2] their availability in the knowledge base is tested.

While editing a knowledge base the system is always able to run. If in a context no ordering rule can fire, the system asks for a symptom value which is used in the preconditions of its refinements. We will extend this facility by asking the user what symptom shall be tested and then generate an appropriate ordering rule for the context.

For testing a knowledge base trace- and debug-facilities were implemented. The user is able to set or reset arbitrary symptom values and inspect the state of all formulas and the formula network according to the actual situation. At the moment we are developing a consistency checker for the knowledge base.

The runtime environment is menu-oriented. Additionally the user is able to enter symptom values by mouse-clicking in some graphics of the technical device. The kernel includes an editor for these graphics.

---

[1] So a data-driven updating of the actual situation is possible.

[2] For example symptom instances referenced in formulas.

# 6. Knowledge Acquisition Using Deep Models

Complex technical devices often have a common property: Due to task customization and rapid further developments only few fully identical instances of a given type are built. This makes it difficult to find somebody who is really an expert for the maintenance of this special machine, especially if the type is relatively new. Service technicians troubleshooting such a new machine cannot rely on machine-specific heuristics, but have to use general technical knowledge, a detailed description of structure and behavior of the machine and a general understanding of the functionality of the device to find any faults.

Since diagnostic expert systems are expected to be able to assist in troubleshooting right after the release of a new type of machine they have to use structural, behavioral and subpart information first, too. As time goes by technicians learn more about a special machine type through the faults they are confronted with during their work. This learning from cases has to be embodied in a sophisticated expert system too, giving the need for a knowledge representation that equally suits the causal (model-based) as well as the heuristic (case-based) knowledge. Furthermore an existing expert system should be easily adaptable to a slightly changed machine, e.g. from a new series or with different optional subdevices (cf. [5]).

## 6.1. Causal Knowledge and Deep Models

We will use the word "causal" in this paper to distinguish things that can be traced back directly and with absolute certainty to structure, behavior and function of a single concrete physical device from those that do not, either since they lack a clear correlation to the physical facts (these we will call "heuristic") or since they depend on many instances of a device and are not certain (called "statistic"). Thus causal knowledge in a technical domain consists of all the facts on structure, behavior and function of the device together with information on how connected parts interact and can be considered synonymous to the term "deep knowledge". From the causal knowledge of a device a representation can be build which in this chapter we will call a deep *model*, or just a model.

## 6.2. The Task: Getting the Causal Knowledge into the System

A possible approach to get some of the causal knowledge about a device into the expert system could be to have an engineer (i.e. someone with sufficient "technical common sense", not necessarily an expert) look at the design plans and diagrams of a concrete machine and write down the information in form of failure detection flowcharts which are then translated into rules of different types. The order in the flowchart would be based on a priori failure probabilities and directly influence the ordering and priorities of the rules. When we tried this way of knowledge acquisition for the first application in project MOLTKE, which was an expert system for the diagnosis of CNC[1] machining centers, it turned out that it was quite time consuming, since the process required several iterations and the final flowcharts were by no means complete (with respect to the underlying descriptions). Worse, new machine series or, even more, types required much of that work again, since most diagrams had to be revisited to check for changes (e.g. new components with new failure probabilities or even new failures).

This caused us to build a tool (MAKE = Model-based Automatic Knowledge Extractor) [6] to do the work of that engineer. MAKE gets diagrams (i.e. the structure) of the machine and uses its built-in knowledge of electrical and mechanical engineering (i.e. its "technical common sense") - e.g. the behavior of typical components - to form a (deep) model of the machine and to derive rules for test

---

[1] Computerized Numerical Control

suggestion and determination of yet unmeasured symptoms (shortcut rules) from this model. Furthermore MAKE is able to organize the rules into contexts and to create corresponding preconditions for them. We call such an expert system without any machine specific heuristic knowledge *basic expert system.*

An important point is that the diagrams can be entered by anyone; there is no need for the special knowledge of an engineer to do that. An engineer (the expert) is needed only to provide information about the typical atomic parts in the descriptions, e.g. valves, switches, contactors, hydraulic pistons. Some statistic knowledge about a priori failure probabilities and additional information on measurement costs have been integrated into the model to allow a more sophisticated test selection; thus the model is no longer purely causal, but contains other knowledge, too[1].

Note that the application domain of the whole MOLTKE system (and therefore of MAKE) is not as simple as digital circuits, the usual example domain for model-based diagnosis.

### 6.3. Available Knowledge when the First Machine is Built

As heuristic knowledge of a new machine will not be available when the basic expert system is to be built, one has to restrain oneself to the information available at that point of time:

- knowledge about the primitive (and maybe complex[2]) parts (components) used
  - * possible connections (ports), i.e. how the component may be connected to others
  - * behavior, i.e. how the component is expected to react to changes in connected components
  - * misbehavior(s), i.e. what typical faults may occur, how they change the expected behavior and how common they are[3]
- knowledge about the connectivity of the components, i.e. a structure description
- knowledge about measuring costs at certain points[4]
- knowledge about the intended functionality of the machine as a whole, i.e. which set of inputs should cause which set of outputs.

The "birth" of a basic experts system then consists of four major steps:

1) An expert builds up a library of the primitive (and maybe common complex) components that are used. This task needs an expert since it should be done as precisely and completely as possible for the sake of the resulting expert system. Of course the library can be used for many different machines thus an expert is not needed for every single machine type or series.

2) A person with technical skill - at least he/she shall be able to read, understand and reproduce technical structure descriptions - is needed to enter the connectivity of a given machine, i.e. to build up the structural model. He/she will select parts from the library and build them together to more complex ones finally describing the whole machine. A graphical editor that allows to mimic the technical diagrams helps to accomplish this step. Note that this task is independent from the availability of a diagnosis expert if the library (cf. step 1) completely covers all components used.

---

[1] However, all the knowledge (information) used by MAKE is available when the first machine of a type is delivered to the customer, i.e. no heuristic information on the special machine type is used.

[2] Often information will be available on more complex subdevices if they have already been used before in other machine types. However, the relevance of this information is not always clear (and is therefore subject to be changed by a mechanism to adapt it from e.g. cases).

[3] This information usually comes from the manufacturer of the primitive component. However, the a priori failure rate can vary dramatically with the position of the part in the machine. Learning from cases can adjust either the heuristic information embedded in the model or the diagnostic system itself to such special situations.

[4] All complex machines have sensors etc. attached to them and provide designated measuring points where data can be gathered inexpensively.

With a fully computerized design, developement and production system (CIM[1]) it should be possible to get the connectivity information immediately from the design data.

3) The designer of the machine can specify the desired functionality in form of input/output behavior. While this information can be left out (MAKE could simulate all input/output behavior from the given knowledge itself) it is helpful to cut down all possible (and maybe unintended or unused) behaviors to the intended and used ones. When dealing with machines with a control component, e.g. a CNC, the CNC programs give information on what functions the machine is expected to perform.

4) Finally, the MAKE system transforms the model of the machine (put together from the library and the connectivity) with help of the desired input/output behavior into a knowledge base for the expert system shell MOLTKE.

### 6.4. The Model

We use a component oriented, hierarchical qualitative model of the machine. What is considered an atomic component is determined by the granularity of the used diagrams and the exchangeability of the parts (e.g. since relays are completely replaced if found faulty, there is no need to model the parts of a relay). Information on typical *primitive* (atomic) components like valves, switches etc. is stored in a library. Using these parts, new assembly groups (called *complex* components) can be built and so on; this way a hierarchical model is built up where the top component represents the entire machine. The general description of a component is found in the component class, its instances represent the concrete components. Connections can be internally modeled as components, but need not to be entered explicitly, since they can be added automatically (e.g. if the user connects two hydraulic ports the system automatically assumes a pipe between them).

A component class stores the following knowledge:

- name of the component type
- ports to other components (optionally with test costs)
- possible internal states (optionally with test costs)
  Internal states can be made available to other components using them as ports.
- behavior of the component
  The behavior is given either in form of tables or by rules: the if-part contains predicates on port values and states that allow to conclude the value of some other port or state in the then-part. These tables/rules represent the constraints the component sets up between its ports and states. Note that components do know about their function, as the "no-function-in-structure" principle[2] is deliberately abandoned to enable the generation of better causal rules from the model.
- subparts and their interconnections (only if the component is non-atomic)
- typical malfunctions with name and effects (if available)
  These typical malfunctions model the behavior of the component when a common failure occurred and enable the system to reason faster. There is always the possibility to fall back to the total suspension of the constraints of the component ("unknown failure"), which is the usual approach of model-based troubleshooting[3] [7].

---

[1] Computer Integrated Manufacturing

[2] This principle is often demanded in the literature on model-based reasoning and qualitative simulation. Its generality, however, while being useful in the prediction of unknown behavior, disallows us some interesting conclusions that we can draw since we already know the function of the part.

[3] Note, however, that shortcut rules will only be 100% correct if the malfunctions listed here are the only ones possible. If "unknown failure" is among the possible misbehaviors of a component class, no certain shortcut rules

- a priori probability of failure (if available)

An actual component is an instance of its class which additionally knows its name, location, neighbors (i.e. which of its ports is connected to which port of which (possibly) other instance), names of its subcomponents (if any) and position in the diagram. Note that each port of each component instance later becomes a symptom in the basic expert system generated by MAKE.

Component classes are immediately cross-checked: topology, states and behavior must be fully specified, all ports of subparts must be connected by connections of the correct type[1] and states must appear in the behavior. When actual parts (instances) are entered, they are cross-checked too to ensure that every subpart class exists and is of proper type and that the connections to the neighbors are ok. Currently a special editor to support the creation of component classes and instances is under development together with a graphic tool to enter the structure of the machine.

### 6.5. Functionality Description

The expected functionality of a device has to be entered as relations between its input and ouput ports. When dealing with computerized numerically controlled (CNC) machines (e.g. MOLTKE's first application: CNC machining centers) even more than these relations can be taken from the CNC programs. These programs do not only provide us with input/output information but usually also contain data on failures the CNC is able to detect. When the CNC detects such a failure it stops the machine and displays a failure number that provides us with first information about where the malfunction happened.

These failure messages are issued if some feedback signal to a started machine order misses or is wrong (e.g. a limit switch did not operate within a given time)[2]. This information allows to focus the attention to a certain part of the machine and is therefore a valuable starting point for the diagnosis. Each failure is associated with a certain machine subcycle that performs a special operation starting with some control system output signals and ending with some feedback input signals. It is possible to conclude from the model which valves, switches etc. are related to these IO signals and are therefore under suspect.

When no CNC is used MAKE has to simulate the device itself to be able to detect deviations from the expected input/output relations and to cross-check the correctness of the functionality specified by the designer. In these cases important inputs and outputs have to be measured before the focus of attention can be moved to a certain part of the machine.

If no functionality at all is specified MAKE must generate all possible input combinations and simulate them through the model to get the respective expected output values. The input/output relations thus generated can than be used as if entered by the designer.

#### 6.5.1. Building Contexts from the Model

The best way to choose contexts is to ask the experts; but since we want to build the basic causal expert system without their help, we have to build the contexts ourselves. Naturally, automatically generated contexts may be not as good as those from the experts, but our experience with the manually chosen contexts showed some typical patterns that will most often fit. Each context (intermediate diagnosis) leads to several more special diagnoses that are its subcontexts. Leaf

---

can be generated from the model since it is impossible to make any conclusions about the behavior of the component. Of course, a case based reasoning mechanism still can derive uncertain shortcuts in such a situation.

[1] Ports have types and directions to ensure proper modeling.

[2] This shows an additional advantage of diagnosis of a CNC-controlled machine: The CNC observes the inputs and outputs of the machine automatically, so we already know these values when the diagnosis starts.

contexts correspond to final diagnoses. An example for a part of a context heterarchy is given in figure 3.

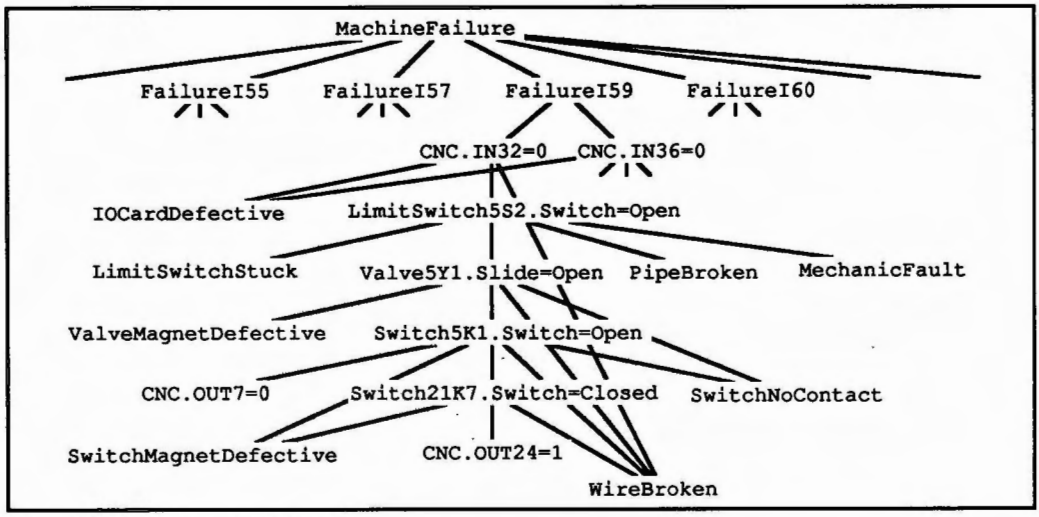



Figure 3: Example of a context heterarchy

The most abstract context and root of the context heterarchy is a context called `MachineFailure`. Its first level of subcontexts is induced by the possiblities of how the input/ouput relations given in the functionality description can go wrong. When a CNC with a integrated diagnostic system is present, this is the level of control system failure messages, i.e. each failure number has its own context (e.g. `FailureI59` in fig. 3). Each of these top level failure contexts now gets subcontexts of its own for each (control) system input that could have been responsible for that failure (e.g. `IN32=0`). Since the same fault could lead to different top level failures, depending on the machine cycle it occurred in (i.e. the inputs), the contexts are organized in a heterarchy instead of a hierarchy to avoid multiple instances of the same context.

MAKE builds a context class C for each possible faulty value V on outport O of component class Co (primitive and complex). The name of C is automatically generated as "Co.O=V" (e.g. `ValveSelection.Switch=Open`). It may seem as if this method creates a very large amount of contexts, but since contexts are only generated for those outports and values that actually force faulty output values in the whole machine, not all combinations will be created. The number of contexts generated thus depends not only on the number of different components and the number of their outports and value ranges, but especially on the constraints in the expected input/output relations given in the functionality description.

There are three basic strategies how to decide from the model what subcontexts a given context gets; each of them constructs a precondition for them that is both necessary and sufficient to establish the diagnosis the context represents:

1) *hierarchical strategy*

This strategy works strictly top-down according to the hierarchical model of the machine, i.e. it first makes sure that the fault occurred inside a component Co (by testing that Co's inputs are as expected while at least one of its outputs is not, e.g. outport O has the unexpected value V) and then switches to the context C for Co. The precondition for entering C thus becomes "all inputs of Co have expected values and output O has unexpected value V". All contexts of the form "$SCo_i.O_{i,j}=V_{i,j}$" become subcontexts of C, if $SCo_i$ is a subcomponent of Co and value $V_{i,j}$ at $SCo_i$'s outport $O_{i,j}$ can cause value V at Co's outport O.

2) *sequential strategy*

Unlike the hierarchical strategy the sequential one does not use the hierarchical structure of the model. Instead, it traces the simulated behavior from the outputs of the machine back to its inputs on the lowest component level thereby necessarily detecting the malfunctioning primitive component. The precondition of a context for the sequential strategy therefore consists only of the fact "Co's outport O has the unexpected value V". All contexts of the form "$Co_i.O_{i,j}=V_{i,j}$" become subcontexts of C, either if $Co_i$ is a subcomponent of Co, $Co_i$'s outport $O_{i,j}$ is directly connected to Co's outport O and $V_{i,j} = V$, or if $Co_i$'s outport $O_{i,j}$ is directly connected to one of Co's inports and a value of $V_{i,j}$ at that inport can cause value V at Co's outport O.

3) *mixed strategy*

This strategy is a mixture of the hierarchical and the sequential ones. It performs backward tracing like the sequential one on a given level, but moves down in the hierarchy only if it can make sure the failure lies within the subcomponent. As we found out this is the strategy the typical service technician seems to follow: He/she traces the faulty output backwards on the top level until he/she comes to a component with correct inputs and faulty output. He/she then moves one level down in the hierarchy and again performs backtracing on that level. The precondition of a context in the mixed strategy therefore is the same as in the hierarchical case, while the subcontexts are selected like in the sequential case.

### 6.5.2. Construction of Shortcut Rules

A shortcut rule tries to establish facts about yet unmeasured port values from already acquired data. Deriving them from a model requires knowledge about the way connected components interact. Shortcut rules derived from the model are total (certain) if all possible failures of components are modeled. Since these rules cannot assume any component to work correctly, too many different possible failures make it impossible to derive any shortcut rule at all. If an unmodeled fault occurs the shortcut rules may conclude wrong port values, but this kind of failure can be found by direct use of the model, e.g. by constraint suspension [7] or similar techniques which, however, will be quite costly. Alternatively, this is also a situation where case-based reasoning can be used.

We have different contexts for the same component with state, one for each possible output value, but the shortcut rules are the same for all of them, since they are only based on the model of the working device. We derive the shortcut rules for a component as follows:

i) Each behavior rule is copied as a shortcut rule *iff* there is no modeled failure that produces an unexpected (different) output[1] on the same inputs.

ii) The inversion[2] of each behavior rule is copied as a shortcut rule *iff* there is no modeled failure that produces the same output on different inputs.

Inside the context, the shortcut rules of each component are collected together with some rules that state the equality of connected ports.

EXAMPLE:

Given `(currentIn = X) -> (currentOut = X)` as the behavior rule for `Wire` and `(Broken (currentOut = 0))` as the failure of `Wire`, we can conclude the following:

---

[1] Outputs here are ports or states that appear on the right side of a behavior rule; inputs appear on the left side.

[2] By inversion we mean the rule that is created by swapping the precondition and the conclusion of the original rule.

Since for all X≠0 `Broken` produces a different (unexpected) output (i.e. 0) on input X than the behavior rule, i) can only be used for X=0. Thus i) leads to the following shortcut rule:

```
(currentIn = 0) -> (currentOut = 0)
```

`Broken` produces output 0 for every input, not just on 0. Thus, according to ii), we must exclude X=0 from the inversion of the behavior rule and can conclude the shortcut rule

```
(currentOut = X) (X ≠ 0) -> (currentIn = X)
```

In the last two sections the question may have arisen why we first model the component with its correct and faulty behavior and then mechanically derive contexts with preconditions and shortcut rules out of it instead of writing them down immediately. Beneath the fact that the representation of the connectivity would be more difficult and that consistency checks would have to be reduced the chosen representation is much clearer, easier to produce and can be used for other tasks too (e.g. construction of ordering knowledge - see next section).

### 6.5.3. Construction of Ordering Knowledge

Our augmentation of the model with failure probabilities and measurement complexity allows us to produce a list of applicable tests ordered according to some chooseable criteria for any situation. To evaluate all possible testing points inside the component a context belongs to (i.e. all the ports of the component and its subcomponents) we build up a dependency graph containing the expected values of these testing points as well as the already measured data[1]. If measured and expected value of some port (node) inside the graph agree, all testing points downward from there can be discarded as long as there is no other connection between them and the root of the graph. If expected and measured value differ this point (node) becomes the new root of the graph (thus cutting all testing points above it) since we found a port nearer to the machine inputs where things went wrong[2]. As can be concluded from this, internal nodes of an existing graph only contain yet unmeasured ports (testing points).

How do we select the next measurement from a given graph? First we examine the information gain we would get if we knew the value of an internal node. This information gain is measured by looking at the resulting graph after each possible outcome of the measurement. The resulting graphs are weighed with the a priori failure probability of the components they include and the resulting value is set off against the difficulty to actually measure the value. How exactly the weights are is determined by the chosen strategy, e.g. whether lots of easy measurements are preferrable to only few but difficult ones. An optimal measuring point would be easy to measure and would split the graph into two equally weighted halfs.

## 7. Knowledge Acquisition Using Empirical Cases

### 7.1. Motivation and Overview

As the task of an expert system which builds up on the MOLTKE toolbox is to simulate the diagnostic problem solving behavior of the respective expert, it is a necessity to represent at least the dynamical parts of the expert´s problem solving behavior which have a great influence on his/her problem solving performance. Especially his/her learning behavior - with respect to the diagnostic task, of course - is of great importance here. The pragmatic solution within the MOLTKE project is,

---

1 Root of this graph is the outport O of the component that showed the wrong value.

2 If we would allow multiple faults, we could not cut the graph this way.

as a first approximation, to distinguish between two basic learning strategies of the expert. The motivation for this is a real world application´s point of view. Therefore it is sufficient for our purposes here. The expert´s basic strategies are:

- *Learning from examples*

  Examples are a kind of problem solving traces which form the expert´s problem solving experience. The expert learns while memorizing such examples and using them as a positive or negative feedback for enhancing his problem solving knowledge.

- *Analogy*

  The expert uses analogies between the given actual problem and known examples to guide and focus his problem solving process. During this process the expert learns new and/or better examples.

Both abilities (in regard of the given task) are acquired during the expert´s normal work. The goal within the MOLTKE project is to utilize both techniques for solving complex problems. The system therefore has a component that memorizes the "learned" examples[1] (which we call cases) and applies analogical reasoning techniques for solving problems based on these concrete cases. The underlying basic hypotheses of our approach are that learning by memory adaptation and analogical problem solving are fundamental techniques which experts (and, as we believe, human beings in general) apply during problem solving. Thus, using them as the system´s central mechanisms maximizes its transparency with respect to the expert´s learning behavior, increases the user´s acceptance concerning the system and decisively improves the knowledge acquisition support.

In this connection the MOLTKE project concentrates on two points of emphasis. The first one is the direct interpretation of the learned cases which allows the application of analogical problem solving as the system´s central mechanism. For further details see [8,9]. Secondly, we stress the usage of an adaptation and extension of the determination-based analogical reasoning approach of [10]. Determinations[2] are similar to MOLTKE´s shortcut rules in the sense that the latter are compilations of analogical inferences which are justified by determinations. Thus, from a simplifying point of view a shortcut rule (if *situation1* then *symptom2* := *value*) can be interpreted in such a manner that situation1 *determines* symptom2. The latter approach enables analogical inferences which are justified with certain restrictions, as the main problem is the acquisition of such determinations/ shortcut rules. Our answer is the automatic generation of total shortcuts from the deep model of the underlying technical system (by the MAKE system, see chapter 6) and of partial shortcuts from the empirical case base using explanation-based learning (by the GenRule[3] system).

The next section gives an overview of the representation(s) of empirical knowledge in MOLTKE and its (respective) principal organization. The generation of partial shortcut rules based on an algorithm presented in [11] is discussed in 7.3. Then our approach of extracting ordering information from the empirical cases is introduced. We complete this chapter with a description of the kernel system´s capabilities for the handling of partial shortcut rules.

## 7.2. Representation of Empirical Knowledge

In MOLTKE empirical knowledge is represented by a collection of examples which we call cases. The respective knowledge sources could be documentation material like service reports, test bench protocols, and test cases or the expert´s remembered experiences. Cases are not only well suited for the representation of such knowledge, but also for the modelling of vague information. In case of

---

1 Learning by memory adaptation, i.e. storing and updating of individual experiences and statistical information.

2 For a definition of determinations within MOLTKE see [11].

3 Generator of empirical MOLTKE Rules

the application on CNC machining centers this affects relations of machine faults to symptoms which depend on smelling, hearing, seeing, feeling, or the geometry of workpieces. From an abstract point of view typical cases can be described as follows:

| diagnostic case | strategy case |
|---|---|
| symptom1 = 1 | symptom1 = 1 |
| symptom2 = 0 | symptom2 = 0 |
| symptom3 = 2 | symptom3 = 2 |
| symptom4 = 3 | symptom4 = 3 |
| --- | --- |
| diagnosis1 | symptom5 |

The part above the line is the description of an empirical situation[1] in the form of a list of symptom values. The part below the line is the solution of the case, i.e. the (empirical) result of the described situation. We distinguish between diagnostic and strategy cases. For the first type, the result of the situation has been to make a diagnosis, for the second, to ascertain a symptom. The diagnostic cases are used for support in the classification task, the strategy cases for the selection of tests. As the symptom values within the description of the case situation are ordered, strategy cases can be automatically derived out of the diagnostic cases. Thus, only diagnostic cases must be acquired. E.g., the following four strategy cases can be derived from the above mentioned diagnostic case:

| | | | |
|---|---|---|---|
| --- | symptom1 = 1 | symptom1 = 1 | symptom1 = 1 |
| symptom1 | --- | symptom2 = 0 | symptom2 = 0 |
| | symptom2 | --- | symptom3 = 2 |
| | | symptom3 | --- |
| | | | symptom4 |

Cases are organized in a hierarchical case memory which is an integration of the experience graph of the PATDEX system (see figure 5) and of an adaptation and specialization of the conceptual memory described in [12]. While there exists only one global case memory for diagnostic cases, strategy case memories can be considered as locally defined for each context. They support the generation of ordering rules for the respective context. The diagnostic case memory is the starting-point for the generation of shortcut rules for all contexts of the kernel system. As the direct interpretation of case memories is beyond the scope of this paper, we must refer to [9] where the fundamental procedure has been described in detail. We now focus on the GenRule system and the kernel system's capabilities of handling uncertain information like partial shortcut rules.

### 7.3. Knowledge Base Refinement Using an Explanation-Based Learning Technique

One important task of the case-based component is the generation of (partial) shortcut rules by the GenRule system. Two basic procedures exist for doing this: the first one has been published in [13] and compares diagnostic cases, which have the same diagnosis, with one another. The second one has been published in [11]. It is more efficient than the first one because cases are compared to minimal diagnosis paths that have been reconstructed out of the MOLTKE knowledge base. It uses a hierarchical case memory of diagnostic cases as starting-point. From an explanation-based learning point of view (see figure 4) the MOLTKE 3.0 knowledge base is the underlying domain theory, the

---

[1] All symptoms which are not mentioned in the case description are assumed to have the value *unknown*.

goal concept is to improve the diagnosis, i.e. having to ask for as few symptom values as possible, the examples are the cases and the operationality criterion is the shortened list of symptoms for which still has to be asked, i.e. the removal of possible tests is the smallest grain size of improvement that is wanted. If a case is presented it is compared to the reconstructed diagnosis path which has the same diagnosis. If the case is "shorter" than the diagnosis path, i.e. it is a positive example in the sense of the goal concept, then GenRule creates a new shortcut rule and adds it to the shortcut rule base of the respective context (if the determination factor exceeds a certain threshold). Thus, the improved domain theory is the starting-point for the next case that is to be processed. Therefore the described procedure is a closed-loop learning mechanism. In the case of a negative example, i.e. the case is not shorter then the diagnosis path, the procedure does nothing. The formal definitions, the description of the underlying algorithm, the computation of the determination (or certainty) factor, and the categorization of the shortcut rules according to their attached determination factors are given in [11].

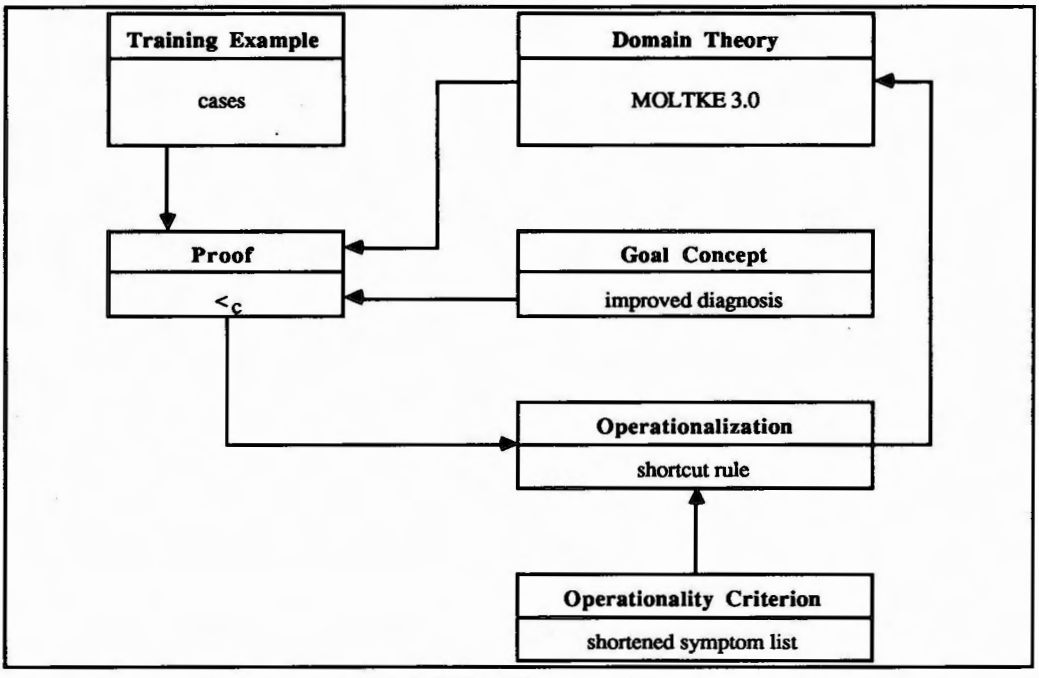



Figure 4: Adapting explanation-based learning for MOLTKE

## 7.4. Extracting Ordering Information from Strategy Cases

Based on the strategy case memories of the respective context the case-based component is able to generate ordering rules for the kernel system. This results in an explicit representation of ordering information which is snapshot-like. An alternative approach is the direct interpretation of the strategy case memory which enables the application of an incremental and closed-loop learning technique. Within this paper we will concentrate on the first approach.

For the extraction of ordering information from the case memory it is sufficient to describe the case memory as in figure 5. Empirical knowledge is represented by means of a weighted directed graph. While the nodes in this graph represent situations, the weights of the directed edges between these

nodes represent the conditional probability of one situation (represented by the end node of the specific arc) occurring next in the diagnostic process under the assumption that another situation (represented by the start node of the arc) describes the current one[1]. Every time the strategy case memory is asked for a new symptom, which should be ascertained, the statistical information represented in the network is used to find out the (more comprehensive) situation which, by prior experience, suggests the best suited symptom to ascertain. This task is accomplished by running a heuristic-driven search through the graph. The result of the search process will be a situation and the additional symptom within this situation will be the one looked for.

The GenRule gives an empty situation to the strategy case memory which answers with a symptom that should be tested. Depending on the answered symptoms GenRule then systematically creates all possible situations which are all given to the case memory. This loop stops when all leaf nodes of the most probable paths are reached. The ordering rules are then built by the association of a situation, given as input to the strategy case memory, with the symptom that is answered by the memory.

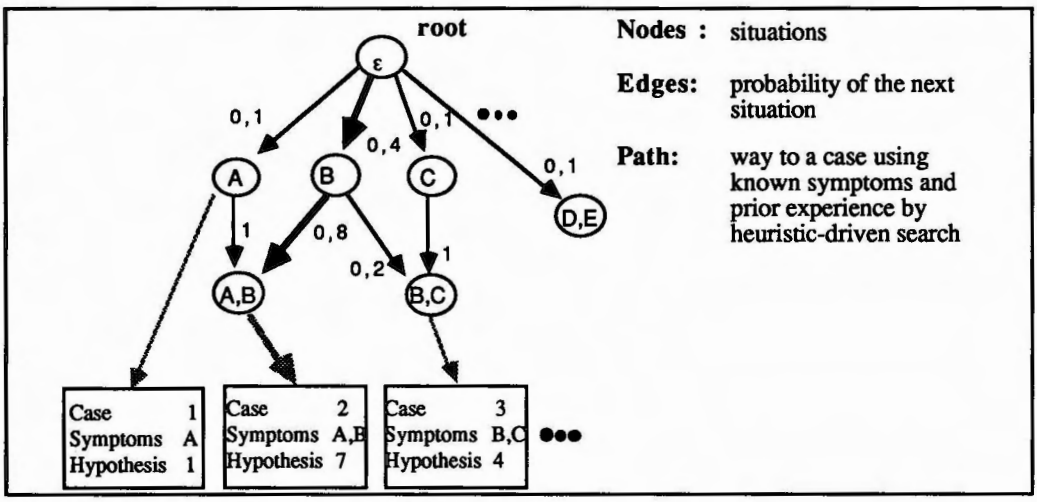



Figure 5: Representation of empirical knowledge

## 7.5. Utilization of Empirical Knowledge in the Kernel System

A basic requirement from a real world application in the domain of technical diagnosis (as in many others) is to make the problem solving process as transparent as possible. One of its consequences is to treat uncertainty in a rather simple manner. Within the kernel system only the partial shortcut rules are uncertain. This uncertainty is described by a list of three determination factors. All factors are computed using the arithmetic expression given in [11] but are related to different sets of cases of the diagnostic case memory. The third value depends on the set of all cases and is therefore the most conservative estimation. The second value depends on the set of cases which all have the same diagnosis and have been responsible for the generation of the considered shortcut rule. This estimation is the most optimistic one. Both values are computed by GenRule. The first value is dynamically computed by the kernel system after a diagnosis has been made. This value depends on the set of all the cases that have the same diagnosis as the actual one, unified with the set of cases

[1] The role of the weights is similar to that of certainty factors for probabilities where the underlying distribution function is not known, too.

that is the basis for the second value. If the actual diagnosis is identical to the one mentioned in connection with the second factor the first and the second factor are equal.

MOLTKE's heuristic-based treatment of uncertainty encompasses the following:

- the kernel system does not compute any probability value during a diagnosis session
- the user can specify:
  - the worst probability category of shortcut rules that is still accepted by the user
  - how many partial shortcut rules are allowed to fire during one diagnosis session
  - how many uncertain symptom values are allowed to be used for getting shortcut rules (including the total ones) fired. This is tested by the shortcut rule interpreter.

If a diagnosis has been made all ascertained symptoms, especially all uncertain ones, are shown to the user. For the uncertain symptoms the corresponding shortcut rules, including their (first) determination factors, are mentioned. If the user states that the presented diagnosis is wrong the system makes several suggestions what to do then:

- change of (arbitrary) selected symptom values
- verification of special uncertain symptom values
- another interpretation of the knowledge base without usage of uncertain information
- another interpretation of the knowledge base with corrected symptom values
- another interpretation of the knowledge base with additional symptom values

As the kernel systems maintains a Rete-like formula network all these actions can be carried out easily. During runtime the system has to remember all uncertain symptoms and to count the number of evaluations of partial shortcut rules and the number of evaluations of shortcut rules with uncertain symptoms. The uncertainty of a symptom affects only the shortcut rule interpreter.

Besides the determination of symptom values partial (and also the total) shortcut rules can be used to define a special ordering strategy, namely to try to ascertain symptoms in a manner that maximizes the number of shortcut rules that can fire.

## 8. The Integration of the Multiple Knowledge Sources

In project MOLTKE two ways of integrating deep and case knowledge into the kernel system are realized. First, compilers can generate the objects used by the shell: this is done in an offline process by MAKE and GenRule (see sections above). For the shell the resulting system is not different from a user defined one. Thus, the whole knowledge acquisition environment of the shell can be used for editing the automatically generated knowledge base. This allows to combine deep modelling with case-based reasoning and common knowledge acquisition techniques.

Second, case and deep knowledge can be used online by different interpreters. We will describe this feature now in more detail.

The default interpreter uses ordering rules as compiled knowledge about strategies. Every context contains an interpreter which evaluates the ordering rule set according to the actual situation. Replacing this context interpreter by a case- or model-based one is easy because of the modularisation of the system. The new interpreter gets the actual situation as input and has to return the symptom instance to test. Thus, the main work is not the integration of the new interpreter into the kernel system but the implementation of the new facility itself.

As described above, we implemented, according to the given specification, an interpreter for model-based ordering knowledge. One possible online interpretation of the strategy case base has been described as the underlying mechanism for the generation of ordering rules. The appropriate modul can easily replace the default context interpreter. The integration of a neural network which was trained with a set of strategy cases is completed. This network gets the actual situation as input and

returns the symptom to test. A detailed description of this approach is not within the scope of this paper.

As the interpretation of ordering knowledge depends only on the context interpreter of a particular context, the system is able to combine the strategy interpreters, i.e. in context A the neural network is used whereas in context B the model-based approach determines the strategy. The use of more than one technique inside a single context would be easy to implement but does not seem to be promising to us.

Besides the described two levels of integration the case- and the model-based component can interact with each other. The case-based component can ask the model-based one for a plausibility check of its cases whereas the model-based part can retrieve statistic information from the case memory.

## 9. Discussion

The starting point for the MOLTKE project has been the end of 1986. It has been a cooperation between our expert system research group at Kaiserslautern (Prof. Dr. M.M. Richter) and the "Laboratorium für Werkzeugmaschinen und Betriebslehre" (WZL) of the Technical University of Aachen (Prof. Dr.-Ing. T. Pfeifer), a world-wide distinguished mechanical engineering institute. The aim of project X6 was to improve the expert system technology for the diagnosis of technical systems. Therefore a complex real world domain (CNC machining centers) has been chosen to allow a deep analysis of the requirements which arise from such a domain and to enable us to develop the proper solutions. Important requirements are: representation and processing of a huge amount of knowledge, sufficient overall performance, ease of adaptability to new sets and types of the technical system, transparency of the diagnostic process, representation and processing of temporally distributed symptoms, flexibility of the diagnostic procedure and automation of the diagnostic process (as far as possible). These requirements are the generalized result of the analysis of the cooperating mechanical engineering research institute. The results have been obtained by a rather complex process and have not been clear in their whole breadth and depth at the beginning of the project. Therefore many different prototypes and specialized architectures (up to now: 10) have been implemented to enable experimental evaluation by the engineering institute.

A comparison of these architectures and the evaluation which of the just mentioned requirements are met by the MOLTKE toolbox (and how far) is beyond the scope of this paper[1]. Anyway, expert system evaluation is a hot research topic and there are no commonly accepted evaluation schemes up to now, especially not for integrated toolboxes. However, we will present an overview of our internal evaluation scheme:

- *A comparison of manually and automatically developed knowledge bases*

  This is currently under develoment for the domain of CNC machining centers.
- *An analysis of the transferability to other machines of the same domain (of CNC machines)*

  In cooperation with a mechanical engineering research institute at the University of Kaiserslautern we have successfully implemented a knowledge base for a 3D-CNC measuring machine.
- *An analysis of the transferability to similar domains*

  At the moment we are developing a knowledge base for the diagnosis of driving machines in mining.

---

[1] The basic assumptions and hypotheses, an overview of the domain of CNC machining centers, the arising requirements and necessary solutions, the main results and an overview of the implemented architectures for technical diagnosis will be published in a book, probably at the end of 1990

- *An analysis of the transferability to other technical domains*
  As an example for a quite different technical domain a diagnostic knowledge base for heterogenous computer networks is currently developed.
- *Adaptation of knowledge to new series and types of the same machine*
  This is a special research objective within our project [5].
- *A comparison of the model-based and case-based component with other approaches*
  This has partially been done up to now (e.g. for the case-based reasoner of the toolbox [9]; for the model-based component [6]), but the completion is forthcoming.
- *A comparison of alternative mechanisms within the toolbox*
  In this point we hope to get a deeper insight when all the above mentioned applications are completed.
- *An overall evaluation of the complete MOLTKE toolbox*
  We believe that the successful integration of methods from several complicated areas into one toolbox is the real research progress we can offer. The ongoing applications will help us to show the usefulness of MOLTKE and provide us with further realistic evaluation.

## 10. State of Realization

The kernel MOLTKE 3.0 is fully implemented including MAKE and GenRule. A case-based reasoner (PATDEX) was implemented as a stand-alone system. The integration into MOLTKE 3.0 is currently in work.

Using the MOLTKE toolbox we implemented an expert system for fault diagnosis of CNC machining centers (see [11,14]) and one for the diagnosis of 3D-CNC measuring machine [15]. Currently expert systems for the diagnosis of failures in heterogenous computer networks and for the diagnosis of driving machines in mining are developed.

The MOLTKE system is implemented in Smalltalk-80 on Sun-, Apollo- and HP-Unix-workstations. It runs on all machines for which the respective virtual machine for Smalltalk-80 was implemented including DECstations, MacII, 80386-PCs etc.

## 11. References

[1] J.Breuker, B.Wielinga: Model-Driven Knowledge Acquisition: Interpretation Models, Memo 87, Deliverable task A1, Esprit Project 1098; 1987

[2] Nökel, K.: Temporal Matching: Recognizing Dynamic Situations from Discrete Measurements, in: Proc. IJCAI 1989

[3] Althoff, K.-D., De la Ossa, A., Faupel, B., Maurer, F., Nökel, K., Rehbold, R.: MOLTKE 3.0, in: Extended abstracts of the GI-workshop on knowledge-based diagnosis systems, University of Kaiserslautern, Febr. 1990

[4] Althoff, K.-D., Nökel, K., Rehbold, R., Richter, M.M.: A Sophisticated Expert System for the Diagnosis of a CNC Machining Center, Zeitschrift für Operations Research (32), 1988, pp. 251-269

[5] De la Ossa, A.: Adaptation of Knowledge to Changes in the Physical System, Technical Report, University of Kaiserslautern 1990

[6] Rehbold, R.: Model-Based Knowledge Acquisition from Structure Descriptions in a Technical Diagnosis Domain, Proc. Avignon 1989

[7] Davis, R.: Diagnostic Reasoning Based on Structure and Behavior, in: Artificial Intelligence 24 (3), 1984, pp. 347-410

[8] Althoff, K.-D., De la Ossa, A., Maurer, F., Stadler, M., Weß, S.: Adaptive Learning in the Domain of Technical Diagnosis, Proc. Workshop on Adaptive Learning, FAW Ulm, 1989

[9] Althoff, K.-D., De la Ossa, A., Maurer, F., Stadler, M., Weß, S.: Case-Based Reasoning for Real World Applications, Technical Report, University of Kaiserslautern 1990

[10] Davies, T., Russel, S.J.: A Logical Approach to Reasoning by Analogy, in: Proc. IJCAI 1987, pp. 264-270

[11] Althoff, K.-D., Faupel, B., Kockskämper, S., Traphöner, R., Wernicke, W.: Knowledge Acquisition in the Domain of CNC Machining Centers: the MOLTKE Approach, in: Proc. EKAW 1989, pp. 180-195

[12] Kolodner, J.L.: Maintaining Organization in a Dynamic Long-Term Memory, Cognitive Science (7), pp. 243-280, 1983

[13] Althoff, K.-D., Kockskämper, S., Maurer, F., Stadler, M., Weß, S.: Ein System zur fallbasierten Wissensverarbeitung in technischen Diagnosesituationen, in: Proc. Austrian Conference on Artificial Intelligence, Innsbruck, 1989, pp. 65-70

[14] Pfeifer, T. Held, H.-J., Faupel, B.: Aufbau einer Wissensbasis für Fehlerdiagnosesysteme von Bearbeitungszentren.- VDI-Z VDI-Verlag 10, 1988

[15] Droste, K., Kaul, U.: Ein System zur Diagnose einer 3D-CNC-Meßmaschine, project thesis, University of Kaiserslautern, 1990

# Shelley — Computer Aided Knowledge Engineering

*Anjo Anjewierden*
*Jan Wielemaker*

Department of Social Science Informatics (SWI)
University of Amsterdam
Herengracht 196
1016 BS Amsterdam, The Netherlands
(anjo,jan)@swi.psy.uva.nl

*Catherine Toussaint*

Cap Sesa Innovation
118 rue de Tocqueville
75017 Paris, France
toussain@capsogeti.fr

**Abstract**

This paper provides an overview of an integrated workbench for knowledge engineering (called Shelley). Shelley interactively supports the analysis and design phases of the KADS KBS development methodology.

## 1 Introduction

There exist of wide variety of tools that support various aspects of the knowledge engineering process. Shelley is a software workbench (like CASE tools for software engineering) aiming to support the knowledge acquisition and knowledge engineering from gathering raw data through to the design of the prospective knowledge based system (KBS). This extensive coverage of the knowledge engineering process has been possible by founding Shelley on the KADS methodology (cf. [8], [13]) for the development of KBS.

KADS views developing a KBS as modelling. Activities in KADS have the objective of building and transforming these models, ultimately resulting in the final model: that of a running KBS. Creating and transforming KADS models is governed by a number of rules. Some of these rules are given by the KADS methodology and state what a model should contain. Other rules govern the order in which the models are built as given by the KADS life-cycle model.

An activity in KADS is supported by an *AST* (Activity Support Tool) in Shelley. The input of an AST is a set of data (*e.g.* protocols) and a model created by another AST. The output of an AST consists of printed documentation, and a model. Each AST ensures that

the rules laid down by the KADS methodology hold for the model being edited, and in a sense an AST is the executable description of what KADS allows within a given activity. The development life-cycle is supported by an editable description of the life-cycle for the project and advice and guidance on the sequencing of activities. The direct manipulation user interface of Shelley allows the user to communicate with the most appropriate visual representation of each model and the elements that make it up.

See the list of references for more information about KADS.

## Acknowledgements

Shelley was designed and implemented by Jean-Marc Aumaitre, Claire Bouchet, Philippe Gobinet, Martin Wood, and the authors. Manfred Aben created the demonstration domain. See [10] for a full list of contributors to Shelley.

The development of Shelley has been supported in part by the Commission of the European Communities under contract ESPRIT Project 1098. Partners in this project were STC plc, SD plc, NTE GmbH, Cap Sesa Innovation, KBSC of TRMC and the University of Amsterdam.

This paper is an extended version of chapter 6 in [13].

# 2 Relation to Previous Work

Shelley supports the analysis and design stages of KBS development. The implementation stage is supported by expert system shells and AI programming languages.

Knowledge acquisition techniques other than protocol analysis (for example those based on repertory grids, laddering, machine learning and so forth) are not explicitly part of Shelley. Incorporating such techniques n the Shelley framework is relatively easy.[1]

Shelley is most related to Keats [14]. The main difference between Shelley and Keats is that Keats provides a richer knowledge representation formalism (and is therefore more appropriate for KBS implementation), whereas Shelley provides richer support for the knowledge engineering process.

The design and functionality provided by Shelley is motivated by earlier work on supporting the KADS methodology.

The first attempt [20] was a normative system, where the knowledge engineer is prompted for features of the domain and the system reasons about the answers the user provides. A disadvantage of this normative approach is that it is very difficult to cover a wide range of domains in any depth. The normative approach, however, seems suitable for certain aspects of KADS, such as feasibility analysis and selection of interview techniques.

The second attempt was KADS Power Tools (KPT) [1]. KPT was a toolkit supporting a limited number of KADS activities in a bottom-up fashion. KPT showed that this kind of tool support is useful, but lack of scope and directions for extensions caused KPT to be abandoned. Shelley subsumes KPT ideas in both the user interface and functionality.

[1] Jean-Paul Billault has developed a repertory grid tool that runs as an AST within Shelley.

# 3 Functions

The workbench idea behind Shelley makes the following functions accessible within a consistent user interface framework:

- *Bottom-up support*
  Activities in KADS are supported by ASTs in Shelley. This allows the user to create and manipulate KADS models in a software supported context. Where appropriate, ASTs provide services like consistency checks. A wide range of ASTs are implemented.

- *Top-down support*
  Advice and guidance is provided at the level of tasks to perform and objects (models, documents) to produce. The technical advice and guidance infra-structure is fully implemented. Only a limited amount of advice is currently provided.

- *Preconfigured components (libraries)*
  The KADS methodology provides several reusable tools that are available in Shelley: the library of interpretation models [7] and the design method library [15] are currently provided. The idea of libraries is that the knowledge engineer selects the relevant components and imports them into the domain.

- *Life-cycle support and project management*
  An AST allows the project manager to define the current status of the project through a graphical representation of the life-cycle. The life-cycle can be customised for the domain.

- *Vertical support*
  Using Shelley is made more attractive by extensive facilities for printing diagrams and generating documents. Shelley also provides browsers for the knowledge base and an annotation facility.

Precisely, how these functions should be integrated and controlled is an open question. The first option is to give Shelley the initiative. The top-down support function decides what the knowledge engineer has to do next, and offers the necessary bottom-up support. For this to work, Shelley must have a good understanding of both the process of running a KADS project and be able to estimate the quality of the information entered into the ASTs. Given that KADS is a methodology, and not a rigid procedure, this appears impossible.

The second option is to have a mixed initiative. The user initiates actions and Shelley comments on them as it sees fit. This option may be the best in the long term, but experience with KADS and Shelley has not made it possible to establish how a mixed initiative can be achieved in practice.

The third option is to give the user the initiative. The user decides on the activity to perform. Shelley can assist on a local level by pointing out syntactical and semantical inconsistencies. Syntactical inconsistencies are violations of the internal definition of KADS models and their constituents (*e.g.* violating the rules of the notation for inference structures). An example of a semantical inconsistency is a **match** knowledge source with only one input metaclass.

In Shelley, the initiative is with the user at all times. Each of the functions listed above is made available on the user's request.

# 4 Scenario

This section gives a scenario of how Shelley can be used in the KADS context. The main purpose of this section is to place emphasis on the requirement that a knowledge engineering activity is ideally performed by running several tools concurrently.

## 4.1 Domain Familiarisation

The knowledge engineer conducts interviews with experts, managers of the client organisation and prospective users of the KBS. The important interviews are transcribed resulting in protocols. The protocols are entered into Shelley using DOTED (see also figure 2). DOTED is responsible for overall management of protocols and types of annotations.

Individual protocols are analysed using a Protocol Editor (PED). While reading, the knowledge engineer notes important terms used in the domain. These terms are entered into the lexicon maintained by LEX. The lexicon can be printed and shown to the expert in order that he can determine its correctness and completeness.

External requirements are also obtained from interviews. External requirements are classified and entered into EXTERQ.

After some study the knowledge engineer identifies pieces of text in a protocol that contribute to the expertise in the domain. These pieces of text (called fragments) are marked and associated with KADS concepts such as static knowledge, concept attribute, knowledge source; or with domain specific concepts. In the former case the knowledge engineer initially keeps an option (*e.g.* associating a fragment with the abstract concept of a knowledge source), later s/he may associate the fragment with a specific knowledge source. In the latter case, the knowledge engineer can associate fragments with domain specific information. For example, fragments describing "causes" of a malfunctioning system can be grouped as required.

This level of annotation gives the knowledge engineer an overview of the domain. The next step is to use this information to construct the domain level. Again the protocols can be used to identify domain concepts.

## 4.2 Domain Conceptualisation

The next step is to create a conceptualisation of the domain. The language KADS defines for this conceptualisation is based on concepts (frames with attributes and values) and concept hierarchies. Concepts also have several predefined attributes enabling the construction of a glossary of the domain. Domain conceptualisation is performed with COND (see also figure 3), and supported by LEX, PED and DOTED.

PED and COND cooperate during domain conceptualisation. Concepts are often identified in a protocol and PED lets the knowledge engineer select the name of the concept and export it to COND. If there is a lexical entry of the same name as the concept, the lexical properties are associated with the concept. COND also warns the user if a similarly named concept already exists (*e.g.* "board" versus "baord").

All ASTs, including COND, provide a facility to view cross references between fragments in protocols and other objects maintained in Shelley. This aids KBS development as follows. The knowledge engineer uses protocols to identify information (by marking fragments in PED and associating them with KADS objects in the DOTED hierarchy). During later

stages of development, the knowledge engineer can retrieve these fragments from anywhere in Shelley. For example, all fragments associated with a given concept in COND.

Hierarchies over a set of concepts provide a static description of the possible inferences that can be made in the domain. Taxonomies are frequently constructed from a small set of well-known relations. The is-a relation models certain types of classification, the causes relation models causal relationships and consists-of models decomposition.

Attributes of concepts allow properties to be represented. These properties can be used to create rules or perform computations in the future KBS.

COND is a dedicated editor for the KADS notation of the domain layer.

## 4.3 Creating the Conceptual Model

The conceptual model is constructed from several parts. The library of interpretation models provides a hierarchy of several interpretation models, the user can browse through. The models have been taken from [7]. Once the user has selected an interpretation model appropriate for the domain, it is exported to the inference structure editor (see also figure 4). ISE is then used to transform the generic interpretation model into a conceptual model for the domain.

Creating the conceptual model involves interactively refining the inference structure and defining the task structure(s). Meta classes are associated with concepts and concept hierarchies at the domain layer (COND) and the data (PED). Knowledge sources are given a type and domain specific name and also associated with their reason of existence (PED). Refining a knowledge source involves creating a sub inference structure, giving a more detailed perspective on the inferencing.

The final output of ISE is a complete and documented inference structure of the domain. Together with the domain layer, the modality analysis (only partially supported by Shelley) and the task layer (not supported by Shelley) the conceptual model is complete. And focus shifts to the design phase.

## 4.4 Design Support

Shelley support for design strictly follows the process and notations given in [15].

Given the conceptual model produced in the previous sections, the knowledge engineer identifies the functional blocks and decomposes these into a functional decomposition. The functional blocks (see also figure 5) also describe functions not part of the conceptual model (*e.g.* I/O and data storage functions). EXTERQ is used to select the functions required by the external requirements.

Next the knowledge engineer creates the I/O and control relations between the functional blocks. For I/O relations data flow diagrams are given, including a description of the data that flows from one functional block to another functional block. The result is a complete description of the functional layer of the design [15].

The Design Method Library is used to select methods given the functions present in the functional layer. The output is an association between functional blocks at the leaves of the functional decomposition and the methods selected to realise these functional blocks. DML suggests the design elements on the basis of the methods selected.

The behavioural layer of the design relates the functional blocks in the functional decomposition to the methods selected in DML, and from there to the design elements.

Finally, the physical layer defines the relation between design elements and modules of the prospective system.

# 5 Requirements

The above scenario hinted at several facilities provided by Shelley. The requirements for these facilities and the way they are made available to the user dictated a number of design decisions in Shelley.

A primary requirement is integration of functionality. It is difficult to predict in what order ASTs are used, which ASTs are used simultaneously and when and how information between ASTs is exchanged. This integration requirement has resulted in a design where all ASTs operate on the same object base: each change on an object is immediately visible to all ASTs. An alternative design, where transformation between ASTs takes place is not desirable for Shelley.

Another requirement is that Shelley must be capable of modelling what KADS can model. The models in KADS are ill-defined (from a formal point of view), but they do impose several requirements on the knowledge representation Shelley must have. KADS concepts (knowledge source, protocol, design element) can be represented as objects (frames with attributes and behaviour). Models can be represented as constellations of objects over relations. An example is the inference structure. This involves three types of objects: *metaclass*, *knowledge source* and *inference structure*. And the following relations:

*inference structure* consists-of *metaclass*, *knowledge source*
*knowledge source* has-output *metaclass*
*knowledge source* has-input *metaclass*

Additional relations are defined to represent the relation between a metaclass and the domain layer.

Finally, Shelley is required to be highly interactive. The details of the user interface and the environment in which Shelley is implemented can be found elsewhere [18], [4], [2]. The two requirements on integration and knowledge representation described above are translated into the design of the interaction between functionality and user interface as follows. Functionality is encoded by *persistent* objects (an executable specification of KADS theory), the user interface is encoded by *visualisers* (windows, diagrams, shapes in diagrams, editors). For each visualiser there is an associated persistent object. A user action on a visualiser is propagated to the persistent object, which in turn informs all its visualisers to update their display. This control mechanism ensures that the contents of the display is consistent with the internal knowledge base at all times (see [18] for a full description and the algorithms used). In addition to consistency, Shelley also keeps a selection server that allows information between ASTs to be exchanged by a mere mouse click.

# 6 Functionality Provided

This section lists the functionality provided by the current implementation of Shelley. More details are found in the Shelley User's Guide [10]. The functionality is listed in the approximate order in which a project would use them, generic facilities are presented first.

## 6.1 Generic Facilities

The following facilities are used throughout the use of Shelley.

**Life-Cycle Editor**
The life-cycle editor (figure 1) allows the project manager and knowledge engineers to keep track of progress in the project. The default life-cycle is derived from the KADS methodology [5], but this can be customised for the domain. The objects manipulated in the life-cycle editor are tasks and documents, where each task has documents as input and output. A thermometer indicates the current status of each task and document. Both tasks and documents can be recursively defined. That is each task can be decomposed into smaller tasks and documents can be decomposed in sub-documents.

**ToolPlaces — Common Working Contexts**
A ToolPlace is a collection of ASTs and their relative layout on the display. With ToolPlaces, the knowledge engineer can either use Shelley from the given set of ToolPlaces or define domain specific ToolPlaces. An example of a ToolPlace (from the scenario given above) is that of "domain familiarisation" which would contain DOTED, EXTERQ and LEX.

**WEB — Visualising Relations**
Each AST defines how relations between objects are visualised. There are two general types of visualisation: connections and placement. Connections are frequently used for data flow, whereas placement (object $A$ inside, object $B$) is used for more structural relationships. The WEB provides an alternative representation of relations by showing a user defined selection of relations that is context independent. The representation is based on small icons for objects, and connections with different textures for different relations. The contents of the WEB can be dynamically explored and layout is fully automatic. All visualisers have an option to create an icon on the WEB. The attributes of an object can be viewed by calling up the Object Browser from any icon in the web.

**Object Browser**
The Object Browser displays a "card" containing all attributes and their values. Similar to the WEB, the Object Browser complements the visualisation chosen by any particular AST. A "Find" option finds all objects matching user entered criteria and displays them in the WEB and/or the Object Browser.

**A+G — Advice and Guidance**
A+G is the advice and guidance tool part of Shelley. The A+G data base consists of cards that contain advice on any topic relevant to the use of Shelley or KADS. The cards are interconnected in a hierarchical structure (supporting a sequential tour through the available advice) and a set of cross-references (supporting the browsing of the advice data base in search of a specific topic). Each card contains advice and a listing of the

references to other cards. The user can activate these references, and the corresponding card is displayed. The types of objects manipulated in A+G are: section (supporting the hierarchical structure), task, document, object (persistent object as manipulated in Shelley) and activity. A special version of A+G allows entering new advice and cross-references, possibly because of specific requirements within the organisation in which Shelley is used.

**Annotations**

Each object in Shelley can be annotated. Annotations can be thought of as the software equivalent of yellow stick-on notes. An annotation contains a piece of text, creation and modification dates and an author identification. Annotations can be created and read using a standardised user interface available from all ASTs.

**Document Generation and Exporting**

The information entered into Shelley is, by default, represented in two forms: graphical (easy to comprehend by the user) or binary (easy to interpret by Shelley itself). The binary representation is also used to store all objects belonging to a domain. Three other types of representation are also required: diagrams, documents and symbolic structures. Diagrams are generated by ASTs using a facility implemented at the level of Shelley infra-structure. Documents are generated by the Document Generation tool. This tool takes an object to generate and a set of format descriptions for the object, its attributes and its relations to other objects. The final result is a piece of text corresponding to the internal object base in a form suitable for processing by a document generation system such as LaTeX. A number of formats are currently provided, for example printing the lexicon and glossary. Additional formats can be defined by textually entering rules used by the document generator. Symbolic representations of the internal object base are useful for exporting to an external system (*e.g.* a programming language or expert system shell). The procedure is identical to the one used by the document generation facility. The implementation of a wide range of output formats has not been achieved, primarily because prospective users of Shelley are bound to have document standards already. Early exploitations of the Shelley workbench have shown that this facility proved itself very usefull.

## 6.2 Activity Support Tools

**DOTED — Domain Text Editor**

DOTED presents a hierarchy of KADS information types. These information types can be associated with pieces of text in protocols (transcripts of interviews, text books and case studies). The user can also extend the hierarchy with domain specific information types. The hierarchy is an aid in structuring the information in a more logical form than can be achieved through the linear structure of text only.

**PED — Protocol Editor**

A protocol is edited in its own PED. PED is both a tool to textually edit protocols, as well as to analyse them. Analysing a protocol proceeds by selecting a piece of text (called a *fragment*) and then associating the fragment with some other object on the display. Initially fragments are associated with nodes in the hierarchy maintained by DOTED. In later stages, fragments can also be associated with objects maintained by other ASTs (*e.g.* concepts in COND, or knowledge sources in ISE). Creating cross-references between fragments and other objects lets the user create a historic perspective on the objects created, and allows maintenance of the KBS in the future. PED

also supports searching for text based on lexical properties and creating lexical entries and concepts by clicking on their name in a protocol. A PED window consists of two sub-windows: one for the text proper and a margin which contains an icon for each fragment associated with some other object in Shelley.

**LEX — Lexicon Editor**

LEX aids in gaining a shallow understanding of terms that are important in the domain. The knowledge engineer can create terms that have a specific meaning in the domain and define lexical properties of these terms. The lexical properties are: translations, synonyms and abbreviations. Lexical entries can be created from PED, where they are most frequently identified.

**EXTERQ — External Requirements Editor**

External requirements are generally noted in the earlier stages of KBS development, whereas they only affect the later stages. EXTERQ supports entering external requirements at any time. Each external requirement has a classification (*e.g.* interface, performance) and a strength (obligatory or desirable). When the external requirement is satisfied by the design of the KBS its satisfied attribute will be set to true. An external requirements finder asks for search criteria (*e.g.* strength, satisfied) and displays all matching external requirements.

**COND — Concept Editor**

COND is an AST in which the domain layer of the four-layer conceptual model is modelled. COND manipulates domain concepts and hierarchies of domain concepts. Concepts are either created by selecting them in PED, or creating them in COND itself. A domain concept has a textual definition as the initial set of attributes. In addition, each concept has an associated lexical entry which defines further default attributes (if there already is a lexical entry with the same name as the concept it is automatically associated with the concept, otherwise a new lexical entry is created when the concept is created).

**IML — Interpretation Model Librarian**

IML maintains the library of interpretation models (as found in [7]) and makes these available for copying into ISE. Functionality includes a diagrammatic representation of the inference structure and a description of the available models.

**ISE — Inference Structure Editor**

ISE lets the user create inference structures from an existing interpretation model (copied from IML). Each knowledge source can be expanded to a sub inference structure, providing a more detailed view on the inferencing process in the domain. ISE is also capable of checking the consistency of an inference structure, using rules about how inference structures are composed and the expected cardinality of the inputs and outputs of knowledge sources. ISE allows a metaclasses to be linked to concepts and concept hierarchies (COND). The concepts are then described by the corresponding metaclass(es).

**TALE — Task Layer Editors**

TALE is only partially implemented. TALE allows the user to define a task decomposition of the domain. The tasks can subsequently be assigned to the user or to the system (see [9]). The tasks assigned to the system have to be described by a task structure (not implemented).

**Functional Decomposition**

The overall output of the ASTs described above is a conceptual model for the domain and a set of external requirements for the prospective implementation. This input is used to create a design. The first step is determining the functional blocks, their decomposition, input/output relations and the control flow.

**DML — Design Method Librarian**

DML contains a library of design elements. The knowledge engineer selects a function type ("problem solving") and DML highlights the possible method classes ("classification", etc.), methods ("hierarchical classification") and the design elements necessary to realise the method selected ("attribute/value pairs", "class definitions" and "classifier"). The design elements are combined in the physical design editor.

**Physical Design Editor**

The physical design editor combines the results of the functional decomposition and the design elements selected in the DML.

# 7 Illustrated Tour

The subsequent pages show Shelley in action. Each page contains a screendump of running Shelley in a demonstration domain. The domain is that of designing mixers in chemical industry (see [19], [6] for a full description of the domain and how KADS was applied).

The user interface specification of Shelley follows the familiar direct manipulation paradigm. Each object on the display (*visualiser* in the Shelley terminology) can be manipulated by the mouse. All visualisers have a popup menu attached through which the knowledge engineer issues commands and mouse gestures (dragging, clicking) are used as accelerators for frequently used commands.

At the top of each screendump is a row of icons. Each icon represent a tool within Shelley (an AST). The icons are arranged in the approximate order in which they will be used in the tour. All ASTs can be used either independently or in conjunction, this largely depends on the domain at hand and the activity the knowledge engineer wants to perform.

# 8 Conclusions

Shelley is a workbench for KBS development following the KADS methodology, much like CASE (Computer Aided Software Engineering) tools support conventional software engineering methodologies. Shelley is different from expert system shells and AI programming environments in that it does not provide support for implementing the prospective KBS, but aids in the analysis and design necessary prior to implementation. An interesting area of further work is creating ASTs that construct prototypes of the KBS from the knowledge entered into Shelley. Shelley is different from many existing knowledge acquisition systems in that it does not depend on any particular knowledge acquisition technique. The architecture of Shelley is such that new techniques can easily be integrated with the core Shelley described above.

The current implementation of Shelley is at the level of a research prototype. The system is fully documented [10], the source code is well organised, but Shelley has not been fully evaluated. Shelley has been used for the analysis for a KBS in the area of insurance [11, 12].

Shelley is written in PCE/Prolog [3], [17] and runs under the SunView windowing system on Sun workstations. Academic institutions in possession of a PCE licence can obtain an evaluation copy.

# References


[1] A. Anjewierden. Knowledge Acquisition Tools. *AI Communications*, 0:29–38, 1987.

[2] A. Anjewierden and J. Wielemaker. An Architecture for Portable Programming Environments. In *Proceedings NACLP'89 Workshop on Logic Programming Environments*, pages 10–16, Cleveland, Ohio, October 1989. Center for Advanced Computer Studies.

[3] A. Anjewierden and J. Wielemaker. Extensible objects. Esprit Project 1098, Technical Report C1.6, SWI, University Of Amsterdam, March 1989.

[4] A. Anjewierden and J. Wielemaker. PCE 3.6 User's Guide. ESPRIT Project 1098 Technical Report UvA-C1-TR-005b, University of Amsterdam, May 1989.

[5] S. Barthélemy, G. Edin, E. Toutain, and S. Becker. Requirements Analysis in KBS Development. ESPRIT Project P1098 Deliverable D3 (task A2), Cap Sogeti Innovation, 1987.

[6] J.P. Billault. Design and implementation of a configuration task. ESPRIT Project P1098, Deliverable E5.2, University of Amsterdam, 1989.

[7] 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.

[8] 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*, pages 265–296, Amsterdam, 1989. North Holland.

[9] 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.

[10] A. Anjewierden (ed.). Shelley User's Guide — Version 1.0. ESPRIT Project 1098 Technical Report, University of Amsterdam, January 1990.

[11] Catherine Toussaint Eric Brunet. A KADS application in insurance. Esprit Project P1098, Deliverable E9.1, Cap Sesa Innovation, 1990.

[12] Catherine Toussaint Eric Brunet. A KADS application in insurance, evaluation report. Esprit Project P1098, Deliverable E9.2, Cap Sesa Innovation, 1990.

[13] P. Hesketh and T. Barrett, editors. *An Introduction to the KADS Methodology.* STC Technology Ltd., Harlow, UK, 1989. ESPRIT Project P1098, Deliverable M1.

[14] E. Motta, T. Rajan, and M. Eisenstadt. A Methodology and Tool for Knowledge Acquisition in Keats-2. In P. Guida and G. Tasso, editors, *Topics in the Design of Expert Systems*, pages 265–296, Amsterdam, 1989. North Holland.

[15] G. Schreiber, B. Bredeweg, M. Davoodi, and B. Wielinga. Towards a design methodology for KBS. ESPRIT Project P1098, Deliverable D8 (task B2), University of Amsterdam and STL Ltd,, 1987.

[16] G. Schreiber, B. Wielinga, P. Hesketh, and A. Lewis. A KADS design description language. ESPRIT Project 1098, deliverable B7 UvA-B7-PR-007, University of Amsterdam & STC Technology Ltd., December 1989.

[17] J. Wielemaker. SWI-Prolog 1.4 Reference Manual. Technical report, SWI, University of Amsterdam, Amsterdam, The Netherlands, January 1990.

[18] J. Wielemaker and A. Anjewierden. Separating User Interface and Functionality Using a Frame Based Data Model. In *Proceedings Second Annual Symposium on User Interface Software and Technology*, pages 25–33, Williamsburg, Virginia, November 1989. ACM Press.

[19] J. Wielemaker and J.P. Billault. A KADS analysis for configuration. ESPRIT Project P1098, Deliverable E5.1, University of Amsterdam, 1988.

[20] B. Wielinga, J. Breuker, and M. van Someren. The KADS system: Functional description. ESPRIT Project P1098, Deliverable T1.1, University of Amsterdam, 1986.

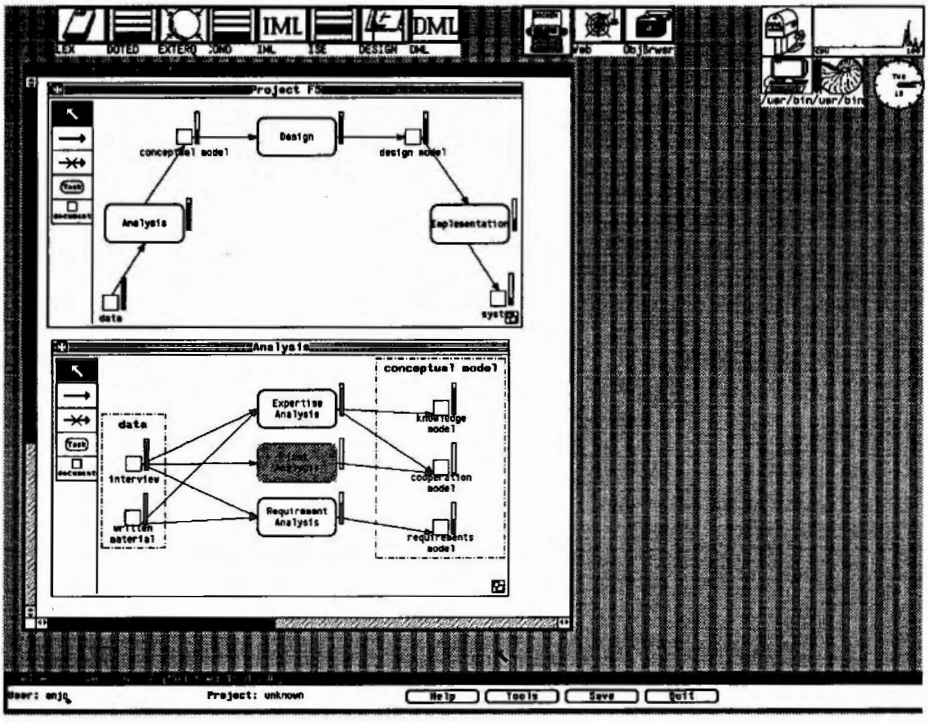



Figure 1: Project Life-Cycle Model Editor (PLCM)

The first screendump shows the top-level KADS life-cycle model (analysis, design, implementation). Shelley currently supports analysis and design. The PLCM lets the project manager customise the life-cycle for the domain at hand. The small boxes are documents ("interview"), the rounded rectangles are tasks to perform. To the left of documents and tasks is a thermometer indicating how much of a document or task is completed. At the lowest level, where no further decomposition is possible, the project manager enters the current state, at the higher levels Shelley computes a weighted average. Tasks that do not apply in a project are greyed out ("task analysis").

The user interface is similar to that of programs like MacDraw. The user selects a mode of operation in the palette on the right, and then points at the operands with the mouse. For example, deleting the input relation between "interview" and "expertise analysis" is achieved by selecting "Delete Relation" mode (crossed out arrow), followed by pointing at "interview" and dragging to "expertise analysis".

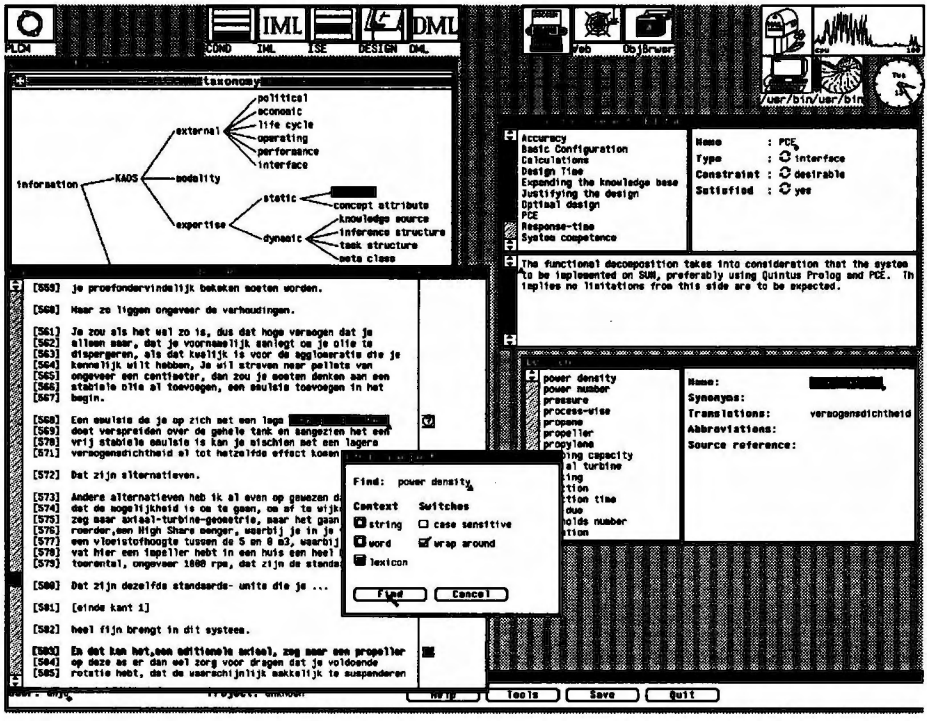



Figure 2: Domain Familiarisation

This screendump contains DOTED (top-left), PED (bottom-left), EXTERQ (top-right) and LEX (middle-right). The knowledge engineer is studying a protocol where the expert is discussing a case study. The knowledge engineer has already identified several relevant terms in the domain and entered them into LEX. The major external requirements have also been identified and classified. The knowledge engineer is now interested in finding relevant protocol fragments that may point to domain knowledge.

The knowledge engineer is interested in the term "power density". In the lexicon he finds the Dutch word for it: "vermogensdichtheid". He starts the find dialogue box from PED, checks **lexicon** mode and PED finds the next occurrence of "power density" and any lexical variant (PED highlights "vermogensdichtheid"; just above the find dialogue).

Next the knowledge engineer can select the fragment and link it to "concept" in the DOTED hierarchy, or create a domain concept.

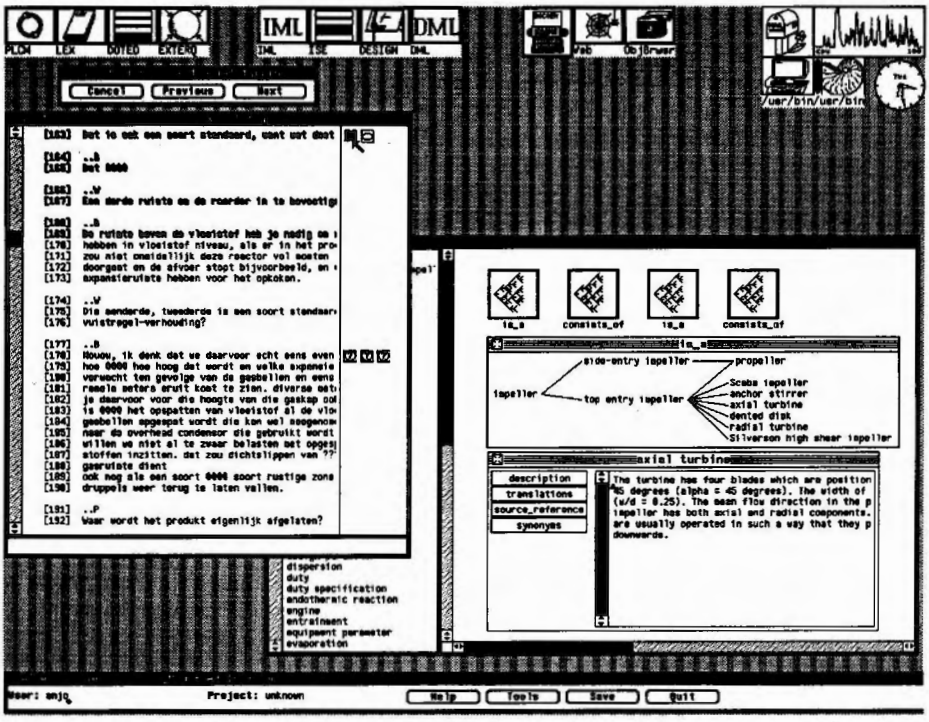



Figure 3: Domain Conceptualisation

After having identified the relevant parts of the domain, the knowledge engineer starts COND (right) to formalise the domain concepts and relations between them.

The screendump shows an is-a hierarchy of impellers, the list of all domain concepts and the description of a particular type of impeller ("axial turbine"). Running PED and COND concurrently, the knowledge engineer can:

- *Define concepts*
  The knowledge engineer links protocol fragments (in PED) to concepts (in COND), and turns fragments into concept definitions.

- *Refer back from concepts*
  Given a concept the knowledge engineer can find all, possibly context dependent, occurrences of a concept in the protocols.

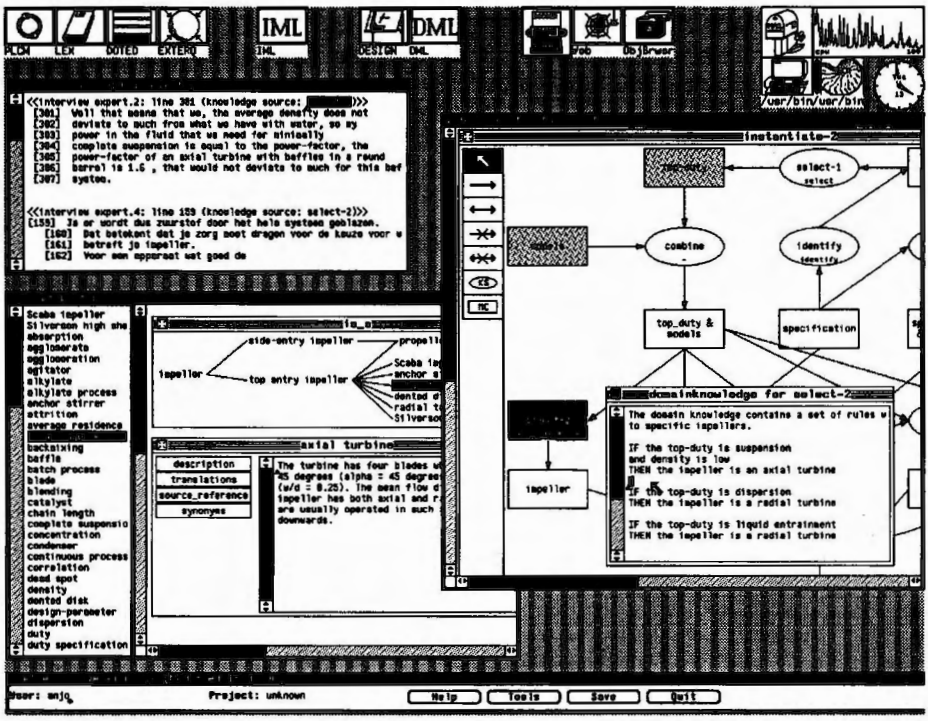



Figure 4: Conceptual Model building

The knowledge engineer has selected an interpretation model from the library (not shown) and now creates an inference structure for the domain. The interpretation model provides the domain independent inferences, whereas the domain concepts and relations provide the domain dependent inferences.

The screendump shows a fragment overview window (top-left) containing all protocol fragments linked to the highlighted knowledge source (is inference function) "select-2", COND on impellers and part of the inference structure (right).

The knowledge engineer uses the fragment overview window, to identify rules for deciding when to use a certain type of impeller. The rule for "axial turbine" is derived from the top protocol fragment.

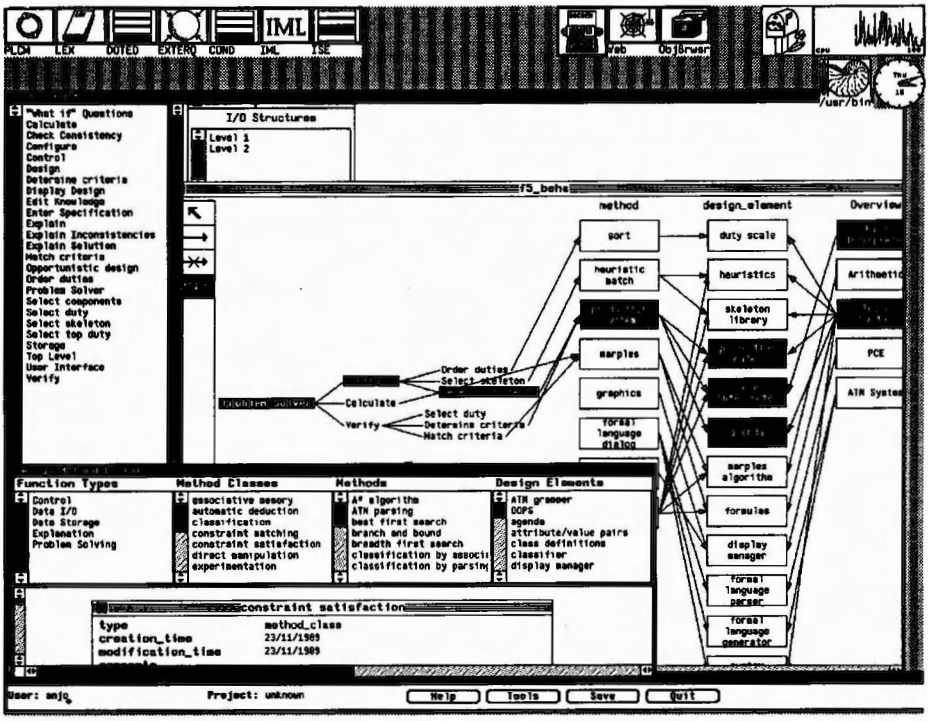



Figure 5: Global Design

The conceptual model now documents expertise in the domain. The next phase is to create a design for the KBS that embodies the expertise. Required input are the conceptual model and the external requirements for constraints imposed by the client organisation. The knowledge engineer defines a decomposition of functional blocks (the decomposition is shown starting with "problem solving"; middle), the I/O and control structures for the functional blocks (not shown).

The DML is used to identify the methods and design elements for the functional decomposition. Methods are selected and linked to leave nodes in the functional decomposition. "Select components", for example, is performed by a production system (as may be apparent from the previous screendump).

The detailed design is not currently covered by Shelley. The theory was not available by the time the specifications for Shelley were agreed. This is unfortunate as the [16] report has also formalised the relation between domain knowledge and KBS design.

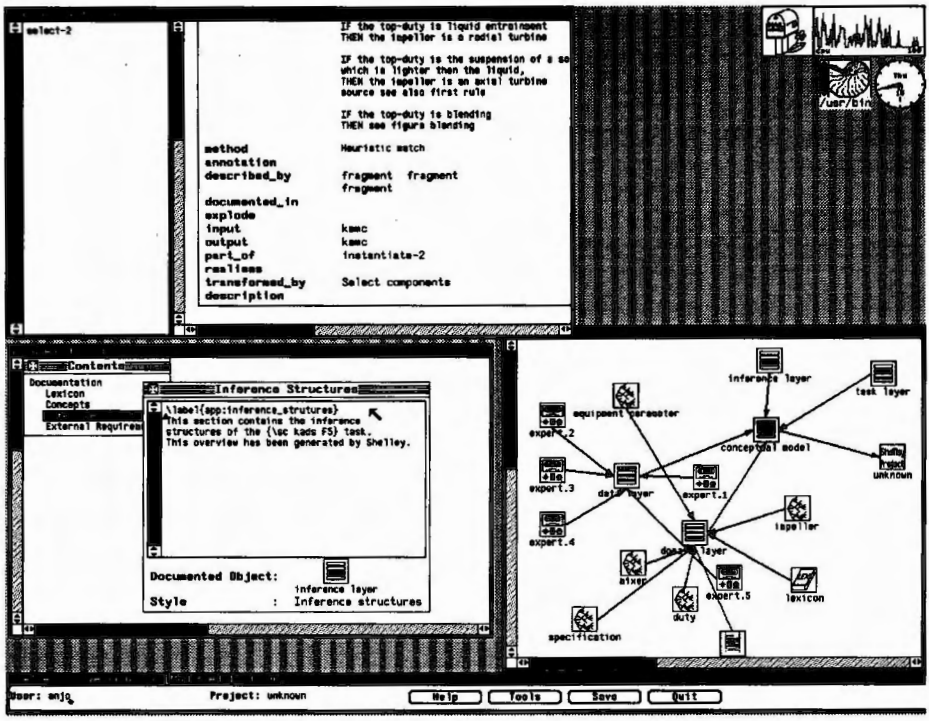



Figure 6: Support facilities

This screendump shows some of the support facilities in Shelley. At the top is the Object Browser. A hypertext facility for browsing through objects as they are stored inside the knowledge base maintained by Shelley. A card for the "select-2" knowledge source is shown. Each field in a card that points to another card is clickable and results in the related card being shown.

The WEB (lower-right) shows various icons of objects and the relations between them. The knowledge engineer can use the WEB to browse through objects by their relation and call up more details in the Object browser. Layout in the WEB is automatically generated.

The Document Generator (lower-left) shows a document frame for the conceptual model. With this document frame, Shelley can at any point during the development process generate a conceptual model document.

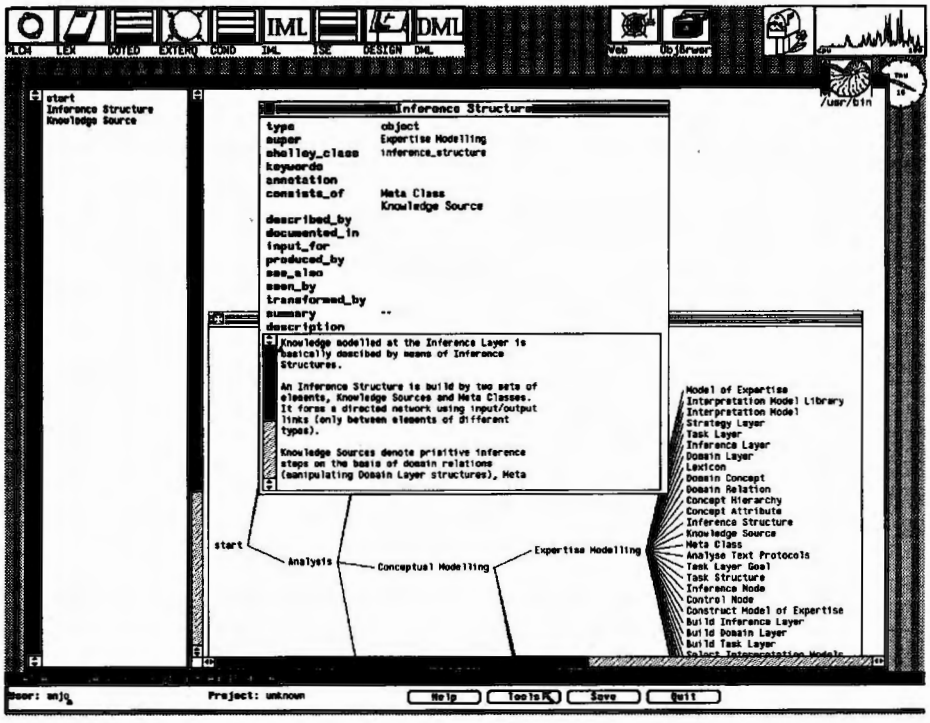



Figure 7: Advice and guidance facility

The final screendump shows the advice and guidance facility. At the bottom is part of the hierarchy of sections. At the top is a card containing more information on "inference structures". The user can browse the advice through clicking.

# StrucTool: Supporting Formal Specifications of Knowledge-Level Models*

John Balder and Hans Akkermans

Netherlands Energy Research Foundation ECN
Software Engineering & Research Department
P.O. Box 1, NL-1755 ZG Petten (NH), The Netherlands
e-mail: balder/akkermans@ecn.nl

**Abstract**

Recent work on second-generation expert systems emphasizes that modelling in knowledge acquisition should be carried out at a conceptual level that abstracts from implementation detail: the 'knowledge level'. As part of this research, formal specification methods are suggested that provide a modular structuring of knowledge-level models of problem solving. We discuss the case for formalization of knowledge models and describe how the development of formal knowledge specifications can be facilitated by their visual representation and manipulation with the aid of the knowledge-acquisition support tool **StrucTOOl** .

## 1 Introduction: Towards Informed and Knowledgeable KA Tools

Knowledge acquisition is increasingly seen as a *modelling activity* [1, 2], whereby engineer-type models are constructed that relate to (restricted) aspects of interest of problem-solving knowledge and that perform certain problem-solving functions. Recent research in second-generation knowledge-based systems [3, 4, 5, 6, 7] stresses that this modelling should be carried out at a conceptual level that abstracts from implementation detail: the 'knowledge level' [8]. This paper takes an additional step by arguing that knowledge-level models must eventually also be expressed in a formal language.

Knowledge acquisition still has many features of an art rather than a science. If it is to become a mature branch of engineering science, it should have as its foundation a *practical knowledge-level theory of problem solving* that is shared by the discipline. The cited research

*The research reported here was carried out in the course of the REFLECT project, partially funded by the ESPRIT Basic Research Programme of the Commission of the European Communities as project number P3178. The partners in this project are the University of Amsterdam (Amsterdam, The Netherlands), the Netherlands Energy Research Foundation ECN (Petten, The Netherlands), the National German Research Centre for Computer Science GMD (St. Augustin, West-Germany) and BSR Consulting (Munich, West-Germany).

on second-generation expert systems might be viewed as an attempt to carry out this program, as it tries to establish a typology of the conceptual elements that recur in models for problem solving. In this context of searching for (what one may call) a theory of 'generic knowledge types and components', we can single out as some of the basic research issues:

- *Generic knowledge components*: Is it possible to identify relatively stable, recurring ingredients of conceptual KBS models that can be used as generic building blocks for knowledge-level models? Here, the work on heuristic classification [9], cover-and-differentiate [10] and, more generally, on generic tasks, role-limiting methods and KADS interpretation models may be mentioned.

- *Formal specification methods*: Can we devise methods for the formal specification of these generic knowledge components, such that their structure and use can be unambiguously expressed at the knowledge level? The knowledge components mentioned above are described using broad conceptual notions and/or in implementation-dependent terminology. Formal descriptions help to bridge this gap. Such an ongoing formalization is a typical element of the historical development of any engineering science. In knowledge acquisition this issue has not yet received much attention — presumably because the purely conceptual aspects have not yet become stabilized — but proposals have recently been put forward by [11, 4, 12, 13].

- *Operationalization in informed and knowledgeable KA tools*: How can such formalized generic knowledge components be operationalized as parts of a knowledge-engineering workbench, to be instantiated for specific applications? This is a generalization of the idea underlying the recent work of [14] carried out in one medical domain. Thus, second-generation knowledge-acquisition tools could be developed that are informed and knowledgeable about a knowledge-level theory of problem solving.

The present paper concerns aspects of the latter two issues. We sketch and comment upon a formal specification method for knowledge models (Sec. 2). It is based upon the KADS approach [1], but is, we believe, sufficiently general and flexible to be of interest to other knowledge-level approaches. Next, we describe our system **StrucT$\mathcal{OO}$l** that is informed (but not knowledgeable) about this formal specification method and thus aids in developing and modifying a formalized knowledge model. Sec. 3 presents an outline of the tool and describes the nature of the support it gives to the knowledge engineer, by visualizing complex knowledge structures and by automatically maintaining various aspects of formal consistency. Sec. 4 discusses the implementation of the tool and shows how elements of formalized knowledge models can be mapped onto object-oriented constructs. An example application is shown in Sec. 5.

# 2 Formalization of Knowledge Model Components

In this section we will put forward some arguments in favour of a formalization of knowledge-level models and consider its role in the conceptual-modelling process. Next, we will give a short overview of the formal specification method as it is used in the **StrucT$\mathcal{OO}$l** system, and make some comments on advantages, weak points and possible variants of the method.

## 2.1 The case for formalization

Although there is no denial on our part of the importance of informal conceptual notions and constructs, we believe that there are a number of good arguments for having formalized knowledge-level models in addition:

1. The terminology used in describing generic knowledge components like 'heuristic classification', 'cover-and-differentiate' or canonical inference steps such as 'abstraction' is diffuse and often ambiguous. What is meant precisely becomes more clear by looking at the computational implementation of such notions, but this is not what one would like to have within a knowledge-level approach. Accordingly, by formally describing conceptual knowledge components *reduction of vagueness and ambiguity* and a better understanding of concepts can be obtained.

2. It seems quite plausible that notions like 'abstraction', 'association', 'refinement' and so on, upon a closer look each represent a family of more or less similar concepts, rather than expressing a single knowledge component. A description in a formal language would provide the possibility for specifying precisely defined variants of a given concept, and hence for achieving a *clearer and more detailed knowledge differentiation* in the context of general schemata.

3. The fact that formal systems have known logico-mathematical properties can be exploited for knowledge engineering. On this account, formalization can be useful to have a *better grip on the properties of a knowledge model*, in connection to, *e.g.*, its computational aspects or reasoning features. For example, it can be demonstrated that a specification in a many-sorted or order-sorted logic rather than in a standard first-order language is a promising technique for solving certain classes of problems (*e.g.*, Schubert's steamroller) [15, 16]. The validity of this argument strongly depends of course on the nature of the chosen formalization.

4. Through its promotion of precision and clarity on the above grounds, formalization *enhances the knowledge-acquisition process* and thus supports the knowledge engineer.

5. This support of knowledge acquisition can be carried a step further if it is embedded in knowledge-engineering workbench tools. Formalization of generic knowledge components constitutes an intermediate step between the qualitative conceptual ideas and the operational constructs in such a workbench. Thus, formalization is of importance as part of a *foundation for second-generation KA tools* that are informed about generic knowledge components.

With the above argument in favour of formalization we do not at all intend to imply that there is no role for informal conceptualizations. On the contrary, these are indispensable. We only want to point out that the knowledge-acquisition process should *eventually* bring conceptual models into a formalized form *at the knowledge level.* There is in our opinion no essential difference here with the role that is played by formalism in other sciences, such as the role of mathematics within physics and engineering. It is also quite imaginable that the formal rigour varies with the field of application, or that a formalization is directed only to critical elements of the conceptual model and hence is only partial. But we do believe that *formal* methods in knowledge acquisition and engineering will be helpful if we want to obtain a real *conceptual* understanding of what is actually meant by intuitive statements of the kind

"these data are abstracted to such-and-such features" or "cover-and-differentiate is a form of heuristic classification".

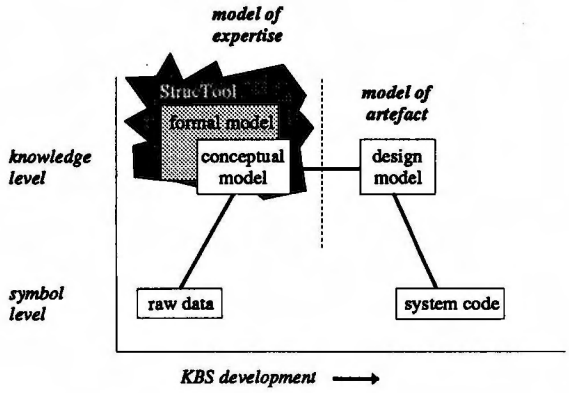



Figure 1: Place of formal models and of the **StrucTOOl** support system in a knowledge-level approach to KBS development

In this paper we will consider in some more detail one possible method for the formal specification of knowledge models, and in particular a system called **StrucTOOl** supporting the development and the modification of such formal specs. In line with the discussion above, the position in the KBS development process of formal models and of the support system **StrucTOOl** may be pictorially represented as in Figure 1 (figure adapted from [2]).

## 2.2 A specification method

Our **StrucTOOl** system is based on the KADS four-layer framework for conceptual modelling [1, 2] and on the related formal-specification method proposed in [11, 4, 12]. We consider the KADS methodology as one of the specific approaches contributing to a theory of generic knowledge types and components, like the work on generic tasks [7], role-limiting methods [6], canonical inference steps as in heuristic classification [9] and components of expertise [5]. The formal-specification method used below has been inspired by the KADS framework but is sufficiently global (probably even too global) to not be restricted to a single knowledge-level approach. As longer papers are in preparation both on KADS in the larger context of generic models for problem solving and on formal specification of knowledge models, we will only give a short discussion here.

The KADS methodology for KBS development promotes a multi-level framework for conceptual models, whereby the various levels or layers express different viewpoints on a body of problem-solving knowledge. Here, the bottom layer contains the static domain knowledge structures. On top of this, one has the inference layer which describes the I/O structure of and the interdependencies between the inferences that are possible on the basis of the domain knowledge. This inference layer might be seen as a 'road map' for the reasoning processes possible in the domain. Next, the inference layer is controlled by the task layer, providing the goal and control structures governing the inference process needed for successfully solving

the given problem. This task layer determines the actual 'reasoning route' taken and thus has a procedural character. However, it is intended only for fixed, routine tasks. Therefore, problem-solving strategies for non-routine situations (such as when an impasse occurs) are expressed in and controlled by the top, so-called strategy layer. This four-layer framework may be viewed as a specialization or refinement of the more commonly encountered distinction made between domain knowledge and control knowledge that describes the use made of the domain knowledge. KADS separates the description of such types of knowledge by positioning them at different layers. These layers are (intuitively) interpreted as a linear chain of meta-levels.

The formal-specification method as developed in [11, 4] contains the following ingredients:

- Each layer is decomposed into *'knowledge modules'*. A module is a small, coherent theory expressed in some logical language.
- Within a layer, more complex theories can be composed by using *module-construction operators* that for example generate the union of two theories.
- The connections between layers are realized by so-called *'lift operations'*, which are in fact a special form of the naming relation between a theory and its meta-theory.
- In addition, inference relations between layers can be expressed by specifying *reflection principles*.

According to this method, formal knowledge models are built from basic entities that are called knowledge modules and represent small coherent logical theories. A module is defined by a two-component specification. First, it consists of a signature giving the names of the sorts in the universe of discourse, the names and schemes of relations and functions and the names of individual constants. The signature provides the alphabet of a logical language. The second component of a module is a set of axioms (knowledge base) in this language representing the theory. Currently, to express theories we use a many-sorted or order-sorted first-order language [17, 15, 16] because, on the one hand, sorts are very useful and natural to conceptually express certain knowledge roles and, on the other hand, such languages have well-known, interesting formal and computational properties.

Now, each separate layer is given by a collection of knowledge modules that are linked in certain well-defined ways. A KADS layer is then seen as a complex theory that is constructed by using special operations on and connections between modules. These connections are inspired by the module-algebra operators developed in the context of algebraic specifications [18, 17]. In building a structured knowledge model a very important operation to support modularity, and thus clarity, of the model is the IMPORT mechanism. With this operator, two modules can be combined into a larger one by adding their signatures and their axiom bases. Accordingly, the import of one theory into another yields the union of these theories.

Another useful mechanism is the EXPORT operator. It defines which part of the signature of a given module is visible for a second module that imports the former one. In this way it provides a form of 'information hiding'. For instance, with the following module specifications:

```
module A          module B
  sorts             import
    S1 S2             A
    ...             sorts
  export              R1 R2
```

**sorts** ...
S3 S4

module B will contain sorts S3, S4, R1 and R2, but sorts S1 and S2 are hidden for this module. The above also indicates the format of a formal specification, which was again inspired by algebraic specification languages. A somewhat similar idea has also been used in the ontological analysis approach by [19].

A next structuring mechanism is the LIFT operation which is a basic construct in connecting different layers. It stems not from module algebra but from meta-level frameworks in logic. Through the LIFT operation it is possible to attach a meta-level constant name (more precisely: a variable-free term) to expressions (axioms, theories) at the object level. Accordingly, it corresponds to what is called a naming relation in logic. The special feature here is that the knowledge engineer has the freedom to specify this LIFT operation, so that it can be made semantically more meaningful than is usual in other contexts. For example, one can in this way formally express that a certain axiom from the domain layer plays at the meta-level (*i.e.*, at the inference layer) the role of a hypothesis that is to be verified. This helps to clearly separate various levels of knowledge.

It appears that sometimes LIFT operations and meta-theories built on top of these do not suffice. This is the case if one wants to express inference relations between a theory and its meta-theory. This can be done by reflection principles in the spirit of [20], which constitute the final construct for the formal specification of knowledge models we want to mention here. They are inference rules that have a formula in one theory as their antecedent and a formula in another (meta-)theory as their consequent. As has been pointed out in a discussion of knowledge-level reflection [21], the so-called enlarged reflection principles [22] form the category of the largest interest to knowledge-based systems.

On the basis of first experiences in the ESPRIT P3178 project 'REFLECT', this formal-specification method appears to be helpful in structuring and in more precisely formulating knowledge models. It can accommodate meta-level frameworks for the modelling of knowledge. Furthermore, since it employs rather general constructs, it has a generic nature and hence is not limited to the knowledge-level approach of KADS alone. The other side of this expressiveness, however, is that in its present state it is too global and not enough specialized with regard to a conceptual theory of KBS problem solving (such as KADS). Also, it currently utilizes the same formal language at all layers. This has a declarative bias, whereas for the task layer it is more natural to have available various procedural constructs. Thus, we suggest that the control knowledge at the task level may be formally expressed in a more natural manner in the language of *dynamic logic*, rather than in some form of predicate logic. Dynamic logic [23] is a multi-modal system that formalizes procedural and control constructs like sequence, truth-testing, selection and iteration. In addition, for the domain layer it is quite conceivable that different languages come into play depending on the domain. For example, in the field of physics a broad class of system structures can be formally specified in terms of a certain directed-graph language [24]. Consequently, the present method seems to be a useful attempt or exercise, but has certainly not yet found a consolidated form. An extensive discussion of our formal framework is given in [12].

# 3 Outline of a Tool for Formal Specs

The aim of the **StrucTOOl** system is to support the construction and modification of formal knowledge models. It embodies and uses knowledge about the formal-specification method as outlined in Sec. 2.

As formal specifications quickly become difficult to handle because of their complexity, an approach similar to *visual programming* was taken. **StrucTOOl** builds two representations of a formal knowledge model: an (invisible) internal one which closely resembles the formal model and an external visible representation. The latter is a metaphor for the formal model. The user interacts solely with this metaphor. User actions are then translated into actions on the internal model. These are effectuated only after the system has checked their validity in the formal framework. On their turn, modifications of the internal representation are reflected in the external visual representation. Figure 2 illustrates this approach. We point out that an internal item can have several visual representations. This is a useful feature due to the fact that in a knowledge model a single knowledge item may play different roles in a reasoning process and, accordingly, the knowledge engineer may want to consider it from different points of view.

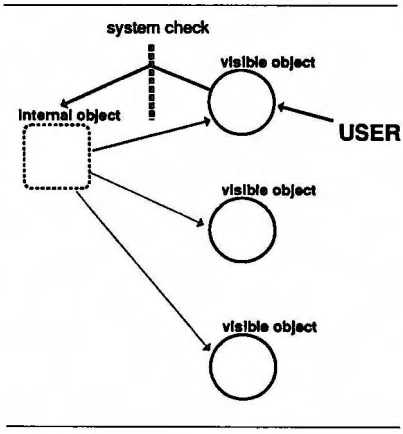



Figure 2: Relations between user interactions, formal knowledge elements and visual objects

A knowledge-acquisition tool for formal specifications should thus provide the following capabilities:

- Transformation of the (partial) specification of a knowledge model into machine terms (here: O-O constructs);
- Visualization of the model specification, including support for different viewpoints;
- Modification of the knowledge model via its visual metaphor;
- Providing support for the modular structuring of knowledge models;
- Continuing checks upon the formal consistency of the knowledge model;
- Transformation of the visual metaphor back into a corresponding formal specification.

**StrucTOOl** provides these capabilities. A further useful feature to evaluate the adequacy of a knowledge model would be to support

- (partial) simulation of reasoning according to the knowledge model.

This might be realized by adding an inference engine (as an O-O method) to the modules. In this way a knowledge engineer may investigate whether the specified knowledge model exhibits the desired reasoning behaviour. Currently, the system does not support this capability, however.

The major functions of **StrucTOOl** are its visual modelling capability and its automatic maintenance of formal consistency. The latter feature comprises a collection of different checks:

- For each module:
  - the sorts mentioned in the schemes of relations and functions must be declared as part of the signature;
  - the same applies to the sorts of constants;
  - the predicate signs used in axioms must be declared in the signature, and likewise the used constants and variables;
  - all names may only be defined once.
- For the exportable part of a module similar criteria are applied, *i.e.*, the exportable part of a module must be consistent in itself.
- The same criteria hold for modules resulting from an IMPORT operation. Name clashes that may occur upon importing a module are treated according to the so-called *origin rule* [18]. This rule says that if the origins of the names, *i.e.*, the modules where the names are defined, are the same, the names are treated as one. Different origins for a name are not allowed.

**StrucTOOl** carries out all these checks and signals errors.

Finally, we note that although visualization of knowledge models already in itself gives rise to a certain reduction of complexity, this is not sufficient. Real-life knowledge models tend to become quite complex. Hence, it is necessary to have means to select or filter the information visible to the user. Preferably, this filtering should be done in terms of the knowledge model, *e.g.*, show all predicates related to a given individual, show all modules that contain a certain individual, show the exportable part of a module or show a knowledge structure at the level of its constituent modules. Our system supports several such possibilities, but further experimentation is needed so as to establish the most useful capabilities.

# 4 Implementation: the O-O aspects

The **StrucTOOl** system was realized using an adjusted version of XLISP[1]. The O-O kernel of this system has the properties of the ObjVLISP model [25]. To add the possibility for visual

[1]XLISP version 1.6, ©1985, by David Betz. Based upon our recent modelling experiences a new version of the system is being developed in Smalltalk.

representation, a separate graphical interface was built using an O-O user interface toolkit called VisualObjects [26].

## 4.1 Representational subsystems

The actual system consists of two parts. One part accomodates the internal representations of the modules and handles all formalism-related actions. Here, each element of the knowledge model is represented exactly once. This part of the system has a global name space similar to the real world one. The other part handles the visualization of the modules in different contexts. Here different representations of a model element may occur, each corresponding to a different point of view. Each module has a local name space. Visualizing a model element means creating visual objects in the corresponding local name spaces of modules. The input of **StrucT$\mathcal{OO}$l** consists of a formal description of modules. The system instantiates a (module) object for each module. Such an object inherits its behaviour from a number of superclasses for, respectively: the behaviour of modules, the behaviour of entities that are part of a module and the behaviour related to visualizing objects. The latter class communicates, via messages, with the visualization part of the system in order to carry out the actual visualization. All module elements minimally have part-of-module behaviour.

In this way a visual representation of complex knowledge models can be realized. The user only has to manipulate the visual representation. The internal formal representation of the modules is then updated by the system. Thus, visual modelling of knowledge structures becomes possible, whereby the burden of maintaining the formal aspects of the specification rests with the system, not with the user.

## 4.2 Class hierarchy

The formal specification method employed here suggests that the following O-O classes are to be present in the **StrucT$\mathcal{OO}$l** system:

- a class for modules, *i.e.*, objects that have the properties of knowledge modules (logical theories);
- a class for the group of items that are part of a module, containing methods concerning the context of items;
- a class for the elements of a logical theory such as predicates and functions;
- a class for mathematical sets.

Moreover, all objects must have some kind of display behaviour. In Figure 3 the class hierarchy of modules is depicted. Note that modules also have the part-of-module behaviour. Thus, nesting of modules to an arbitrary level is possible.

In our implementation we represent each signature by a class that is an instance of the metaclass Signature. This signature instantiates modules with a private instance variable containing the theory. The metaclass Signature contains methods for the manipulation of the signature. Here, all checks on the consistency of signatures, as described in Sec. 3, are performed whenever it is modified. Each signature keeps track of the signatures by which it

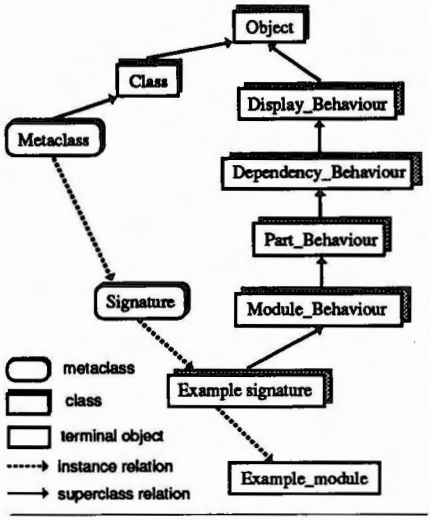



Figure 3: Class hierarchy for module objects

is imported. Thus it can notify these signatures of any changes, so that they can for instance update their visual representation. This notification works recursively.

In the visual representation of the system, signatures are not explicitly visible to the user. The user directs all commands to a module. Several methods in Module_Behaviour are used to delegate messages that are in fact meant for the signature. Further, there are methods for manipulation of the theory of the module. These methods guard aspects of consistency of the axiom base, as described in Sec. 3. Several methods are used to obtain information about the contents of the signature and theory of a module. These methods are invoked by the visualization part of the system.

Operationally, a module is an environment or context for a local theory. It contains the names of items plus logical predicates on these items. An item can be part of one or more modules. For several purposes, *e.g.*, for displaying an item, it is necessary that an item knows about the context it is in. The abstract superclass Part_Behaviour defines an instance variable that contains all contexts of an item. It provides meta-methods for the delegation of messages from an internal object to all contexts in which its visual counterparts 'live'. As these on their turn may have visual counterparts living in different contexts, this is done recursively. All message paths eventually will end in the KADS knowledge-model layers which have a null context.

**StrucTOOl** allows dynamic modification of modules by the user. In order to maintain the consistency of the specification an element of a module must know if other elements depend on it. For instance, a predicate depends on the existence of its arguments. If one of these items is removed the predicate becomes obsolete and must be removed. The superclass Dependency_Behaviour supports two methods, one to add an item to the dependency list of an entity and another one to destroy all items that depend on it. The dependencies are module-specific.

The superclass Display_Behaviour implements the methods to display items. The main method, **Browse**, uses the meta-methods from Part_Behaviour in order to instruct the visualizing system to display an element in all its contexts. **Browse** allows specialization of the display method by invoking the (empty) method **Private_Browse** which can be redefined in its subclasses. Module_Behaviour uses this feature to display the elements of the module.

For each relation scheme that is given in the specification of a module a relation class is made. This class contains the scheme, *i.e.*, the number of arguments of a predicate plus the corresponding sorts. Predicates are instances of such classes. Each predicate has an instance variable containing the names of its arguments. At instantiation time the number of arguments as well as their types are checked against the scheme that is accessible via its class. Also, instance variables are provided for the names of variables and for the body of the clause. Functions are dealt with in a similar fashion.

Finally, there are a number of auxiliary classes. The most important one is Set which implements mathematical sets. A subclass of Set, LinkSet, implements the IMPORT and EXPORT behaviour of modules. Instances of LinkSet are assigned to the instance variables of a signature object that represent the sets of names of sorts, constants, relations and functions of a module. The corresponding axiom base is also represented by an instance of LinkSet. In module algebra IMPORT is an operator that modifies a module by adding to its signature and axioms the signature and axioms of the imported module. This is implemented by adding to each set in the importing module a link to the set in the corresponding imported module, rather than a copy of it. The method **:elements** that delivers the contents of a set, automatically expands all links by sending the same message to the imported module. Note that in this way imports are handled recursively. The EXPORT facility is handled by effectively dividing all sets in two parts: the hidden and the exportable part. The method **:elements** can deliver either the whole set or just the exportable part.

# 5 An Example Application

In this section the features of our tool are illustrated by an example formal knowledge model. The corresponding conceptual model follows the four-layer KADS framework [1], shortly discussed in Sec. 2. We employ an adjusted version of the formalized description in [4] and [27], related to the cover-and-differentiate problem-solving method [10]. The original version of this formalization was created entirely manually. We used this version as a starting point to create an improved model. This was done iteratively by feeding the original specification into **StrucT**$\mathcal{OO}$l and correcting the inconsistencies reported by the system.

All entities in the model are visually represented as icons or windows, depending on how the user wants to look at the system. If an entity is considered atomic the user will choose the iconic representation and the external structure of the entity will be visible. If the user is interested in the internal structure the icon can be opened into a window. In this way the user can easily switch between different levels of aggregation. The menus attached to each icon c.q. window reflect the role the entity plays, *i.e.*, icon menus concern external properties of an entity whereas window menus concern internal properties. Furthermore all menus have an entry for making a hardcopy.

## 5.1 Layer level

The top-level view produced by **StrucTOOl** shows the four KADS layers as icons. Each layer can be opened to show the hierarchy of knowledge modules within that layer. In Figure 4 three layers of the knowledge model have been opened. This figure gives an impression of the structure of the model with respect to the modularisation. (The original system uses colour as a result of which there is some information loss in the shown black and white reproductions.)

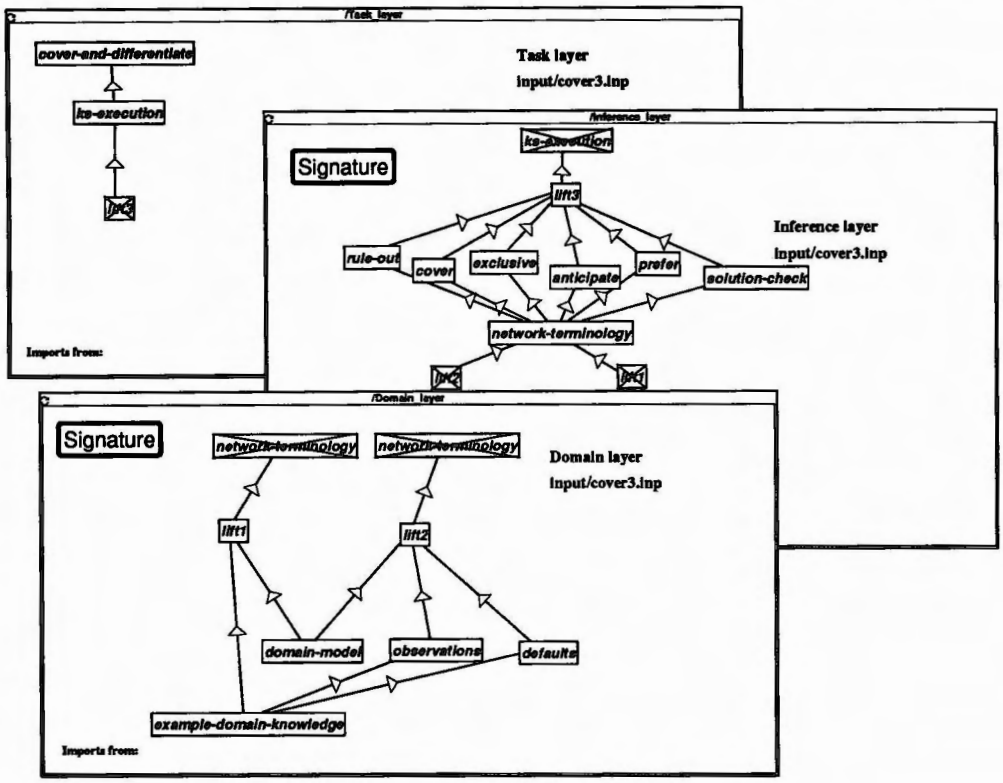



Figure 4: Top-level view of the knowledge model

The arrows show the IMPORT relations between modules within a layer. LIFT connections between layers are depicted by arrows to crossed-out icons. These icons are so-called virtual modules representing modules in another layer. Opening such icons results in opening the real module, thereby bringing up and possibly opening the layer it resides in.

The icon menu of layers gives access to general activities such as saving the formal description of the model under construction. The window menu handles adding new modules to a layer. It is at this level that the user checks the scope of module elements. This scope is determined by the various IMPORT and EXPORT operators in modules and is often very difficult to trace by hand. A menu attached to each module-icon lets the user select a module element for scope tracing. In Figure 5 the trace for the scope of sort *yes-no* in module *observations* has been indicated by presenting the involved modules in black. In the situation depicted here the user has also requested the system to trace the origin of the sort. The icon menu for modules also enables the user to create imports of modules (not shown here), thus

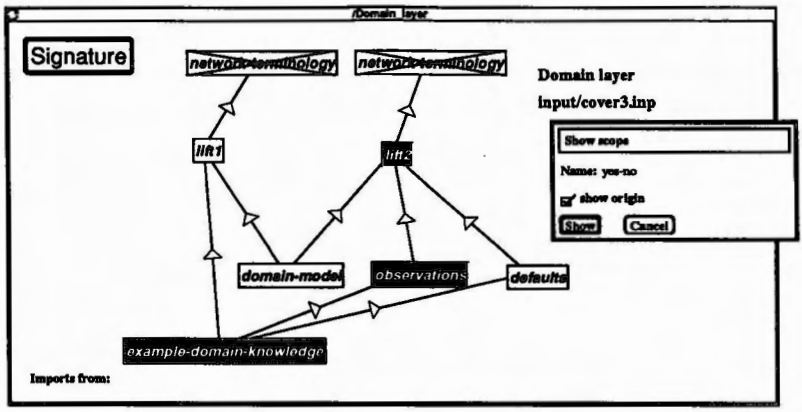



Figure 5: Layer showing the trace of a sort

modifying the external structure of the module and the internal structure of the layer.

## 5.2 Knowledge module level

Next, we zoom in from the layer level to the level of individual (knowledge) modules. Figure 6 shows an opened module that is given by the following formal specification:

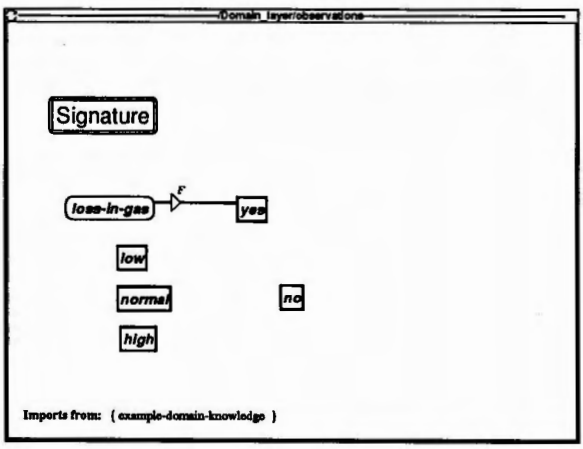



Figure 6: Knowledge module showing individual constants and functions/predicates

```
module observations
% LAYER Domain_layer
% imported_by: (lift2)
  export
```

**import**
example-domain-knowledge
**axioms**
loss-in-gas = yes

The visual representation of modules concentrates on the theory contained in the module, *i.e.*, the individual constants (rectangles) and the predicates related to them (rounded boxes). A simple visual clue (thick border) indicates elements that have been imported from other modules. Arrows indicate the elements of predicates or functions. Also, not visual here, colour is used to distinguish the different items. At the bottom an indication is given of the modules imported by the considered module.

The module window menu coordinates several actions pertaining to the internal structure of the module:

- modification of the signature by adding sorts, relations, functions or constants;
- importing modules;
- modification of the theory by adding predicate and function instances according to the schemes defined in the signature (both additions to the signature and to the theory are either exportable or local);
- highlighting special categories of interest, *e.g.*, exportable elements, elements connected to a selected one etc.;
- general activities such as printing the graphical representation of a module.

All actions of the user are supported by popup menus that restrict actions to those valid within the context of the module. The menus obtain the information they need dynamically from the module when they pop up. For instance, when choosing the type of a new constant, the popup menu displays the available sorts in the module to choose from. Furthermore, it is seen from the examples shown in the Figures that the user can easily switch between different levels of aggregation within the knowledge model. The signature of the module is not directly visible but can be inspected by opening the signature icon as shown in Figure 7.

## 5.3 Theory element level

At the deepest and most detailed level of inspection the knowledge model contains the items that make up the theory in modules, *i.e.*, individual constants and predicates/functions. The icon menu of elements has entries for destroying the element and for showing the connections of the element to other elements. The latter feature is very useful when a module is not yet properly laid out. Imported elements have the same icon menu without the destroy option. Though elements are considered to be atomic, *i.e.*, to not having internal structure, they can be opened. Two elements have been opened in Figure 8:

- an axiom showing its text representation and import path and
- a constant showing its sort and import path.

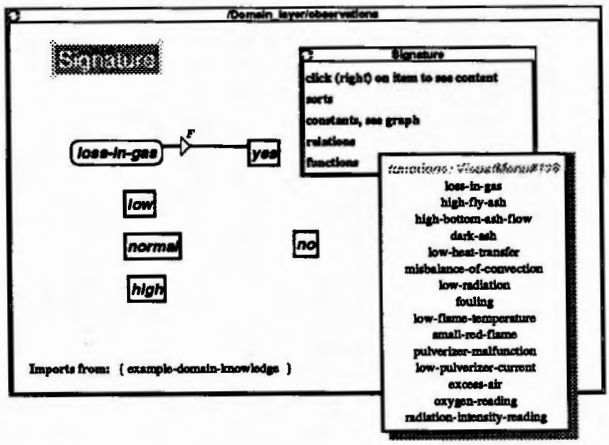



Figure 7: Module with opened signature showing defined functions

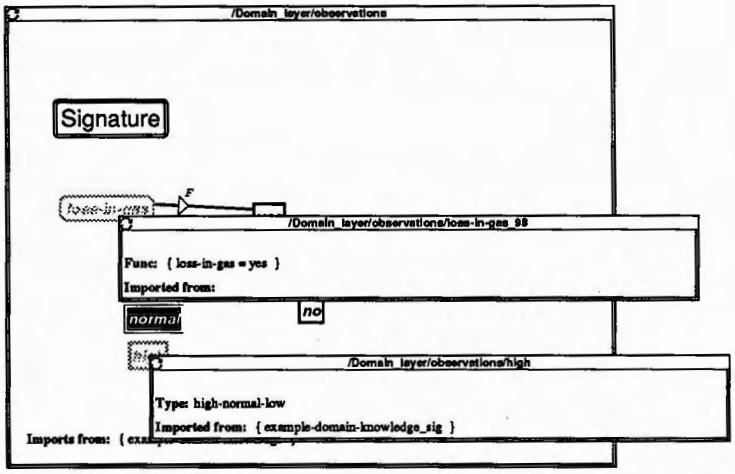



Figure 8: Module showing contents of individual elements

This tool has been used in formalizing several conceptual KADS models. A new implementation based on Smalltalk is being built where the experience with **StrucTOOl** has been taken into account. Important extensions are:

- better support in terms of KADS concepts, *e.g.*, the inference layer has also a subview that depicts the inference model in terms of metaclasses and knowledge sources;
- interactive parsing of modules with explanatory feedback to the user in case of errors;
- forward reference of modules, *e.g.*, with regard to import relations with other modules that are yet to be defined;
- better support for LIFT operators.

# 6 Conclusion

This paper has discussed the case for and the role of a formalization of knowledge-level models. We have outlined a formal-specification method and an associated knowledge-acquisition support tool **StrucTOOl** that facilitates the construction and modification of formalized knowledge models. The system consists of two parts. The first, internal, part links the knowledge model elements to object-oriented constructs, handles the manipulations of the formal model and maintains its formal consistency. The second part provides various visual representations of the formalized knowledge model the user (*i.e.*, the knowledge engineer) can interact with.

In this manner our tool supports the knowledge engineer's activities in the following ways:

- It takes care of a number of formal aspects of the modelling task, so that the user is able to concentrate upon the conceptual aspects.
- It supports a clear separation of knowledge components, for instance by modular decomposition. This feature of modularity fits nicely with the object-oriented approach.
- Through its visualization functions, it makes complex knowledge structures more transparent to the user and facilitates their visual modelling.

**Acknowledgment** The authors are indebted to Wim Rijnsburger (ECN) who implemented VisualObjects and also provided invaluable help in building the graphical interface for **StrucTOOl** .

# References

[1] B.J. Wielinga and J.A. Breuker. Models of expertise. In *Proceedings ECAI-86*, pages 306–318. ECCAI, 1986.

[2] A.Th. Schreiber, J.A. Breuker, B. Bredeweg, and B.J. Wielinga. Modelling in KBS development. In J.H. Boose, B.R. Gaines, and M. Linster, editors, *Proceedings 2nd European Knowledge Acquisition Workshop EKAW-88, Bonn*, pages 7.1–7.15, St. Augustin, 1988. GMD. GMD-Studien 143.

[3] W.J. Clancey. The epistemology of a rule-based system — a framework for explanation. *Artificial Intelligence*, 20:215–251, 1983.

[4] B.J. Wielinga, J.M. Akkermans, A.Th. Schreiber, and J.R. Balder. A knowledge acquisition perspective on knowledge-level models. In J.H. Boose and B.R. Gaines, editors, *Proceedings 4th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, pages 36.1–36.22, Calgary, 1989. University of Calgary.

[5] L. Steels. Components of expertise. AI Memo 88-16, AI Lab, Free University of Brussels, 1988.

[6] J. McDermott. Preliminary steps toward a taxonomy of problem-solving methods. In S. Marcus, editor, *Automating Knowledge Acquisition for Expert Systems*, pages 225–255. Kluwer Academic Publishers, Boston, MA, 1989.

[7] T. Bylander and B. Chandrasekaran. Generic tasks in knowledge-based reasoning: The 'right' level of abstraction for knowledge acquisition. In J.H. Boose and B.R. Gaines, editors, *Knowledge-Based Systems, Volume 1: Knowledge Acquisition for Knowledge-Based Systems*, pages 65–77. Academic Press, London, 1988.

[8] A. Newell. The knowledge level. *Artificial Intelligence*, 18:87–127, 1982.

[9] W.J. Clancey. Heuristic classification. *Artificial Intelligence*, 27:289–350, 1985.

[10] L. Eshelman. MOLE: A knowledge-acquisition tool for cover-and-differentiate systems. In S. Marcus, editor, *Automating Knowledge Acquisition for Expert Systems*, pages 37–80. Kluwer Academic Publishers, Boston, MA, 1989.

[11] J.M. Akkermans, B.J. Wielinga, A.Th. Schreiber, and J.R. Balder. Towards a formal specification of knowledge models. Technical Report ECN-89-006, Netherlands Energy Research Foundation ECN, Petten (NH), The Netherlands, 1989.

[12] J.M. Akkermans, F. van Harmelen, A.Th. Schreiber, and B.J. Wielinga. A formalisation of knowledge-level models for knowledge acquisition. Technical report, Netherlands Energy Research Foundation ECN, Petten (NH), The Netherlands, 1990. Submitted to the International Journal of Intelligent Systems.

[13] T. Wetter. First-order logic foundation of the KADS conceptual model. Technical report, IBM Germany Scientific Center, Institute for Knowledge-Based Systems, Heidelberg, 1990. To be presented at EKAW-90.

[14] M.A. Musen. *Automated Generation of Model-Based Knowledge-Acquisition Tools.* Pitman, London, 1989. Research Notes in Artificial Intelligence.

[15] C. Walther. *A Many-Sorted Calculus Based on Resolution and Paramodulation.* Pitman, London, 1987. Research Notes in Artificial Intelligence.

[16] M. Schmidt-Schauß. *Computational Aspects of an Order-Sorted Logic with Term Declarations.* Springer-Verlag, Berlin, 1989. Lecture Notes in Artificial Intelligence No. 395.

[17] J.A. Goguen and J. Meseguer. Unifying functional, object-oriented and relational programming with logical semantics. In B. Shriver and P. Wegner, editors, *Research Directions in Object-Oriented Programming*, pages 417–477. MIT Press, Cambridge, MA, 1987.

[18] J.A. Bergstra, J. Heering, and P. Klint. Module algebra. Technical Report CS-R8844, Centre for Mathematics and Computer Science CWI, Amsterdam, 1989. To appear in the Journal of the ACM.

[19] J.H. Alexander, M.J. Freiling, S.J. Shulman, S. Rehfuss, and S.L. Messick. Ontological analysis: an ongoing experiment. In J.H. Boose and B.R. Gaines, editors, *Knowledge-Based Systems, Volume 2: Knowledge Acquisition Tools for Expert Systems*, pages 25–37. Academic Press, London, 1988.

[20] R. Weyhrauch. Prolegomena to a theory of mechanized formal reasoning. *Artificial Intelligence*, 13:133–170, 1980.

[21] J.M. Akkermans and M. Reinders. Preliminary comments on knowledge-level reflection in KBS. Technical Report Esprit-project P3178 REFLECT RFL/ECN/I.1/1, Netherlands Energy Research Foundation ECN, Petten (NH), The Netherlands, 1990.

[22] F. Giunchiglia and A. Smaill. Reflection in constructive and non-constructive automated reasoning. In H. Abramson and M.H. Rogers, editors, *Meta-Programming in Logic Programming (Proceedings META-88 Workshop, Bristol)*, pages 123–140, Cambridge, MA, 1989. MIT Press.

[23] D. Harel. Dynamic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, Vol. II: Extensions of Classical Logic*, pages 497–604. Reidel Publishing Company, Dordrecht, The Netherlands, 1984.

[24] J.L. Top and J.M. Akkermans. Processes as components: on the primitives of a qualitative scientific physics. Technical Report ECN-I–90-2, Netherlands Energy Research Foundation ECN, Petten (NH), The Netherlands, 1990. Proceedings ECAI-90, Stockholm, to appear.

[25] P. Cointe. Metaclasses are First Class: the ObjVLisp Model. In *OOPSLA'87 Proceedings*, pages 156–167, Orlando, Florida, USA, 1987. Special Issue of SIGPLAN Notices, Vol. 22, No 12.

[26] W. Rijnsburger. VisualObjects: A server-resident user interface toolkit for loosely coupled NeWS clients. Presented at Sun Expo Europe, The Hague, April 18–20, 1990.

[27] Guus Schreiber. Example formal representations in the analysis domain. Technical Report RFL/UvA/I.1/3, University of Amsterdam, September 1989.

# Capturing Design Knowledge for Engineering Trade Studies

John H. Boose, David B. Shema, Jeffrey M. Bradshaw

Boeing Advanced Technology Center, 7L-64
Boeing Computer Services, P.O. Box 24346
Seattle, WA 98124

**Abstract.** Currently, much of the information regarding decision alternatives and trade-offs made in the course of a major program development effort is not represented or retained in a way that permits computer-based reasoning over the life cycle of the program. The loss of this information results in problems in tracing design alternatives to requirements, in assessing the impact of change in requirements, and in configuration management.

To address these problems, we are studying the problem of building an intelligent, active *corporate memory facility* which would provide for the capture of the requirements and standards of a program, analyze the design alternatives and trade-offs made over the program's lifetime, and examine relationships between requirements and design trade-offs. Early phases of the work have concentrated on design knowledge capture for the Space Station Freedom. We have demonstrated and are extending tools that help automate and document engineering trade studies (the topic of this paper), and we are developing another tool to help designers interactively explore design alternatives and constraints.

## 1.0 Overall Problem

Under NASA contract NAS2-12108, the Boeing Advanced Technology Center is conducting research leading to a corporate memory facility (CMF). A corporate memory facility would provide facilities for capturing and using decision history and rationale throughout a major program's life cycle. This effort is jointly funded by OAST's AI Program and the Space Station Freedom Advanced Development Program.

Currently, much of the information regarding alternatives considered and trade-offs made in the course of a major program development effort is not represented or retained in a way that permits computer-based reasoning over the life cycle of the program. The loss of this information results in problems in tracing alternatives to requirements, in assessing the impact of change in requirements, and in configuration management (Boeing Computer Services, 1989a,b).

There is not an integrated set of capabilities to assist in generating and evaluating or reevaluating program alternatives. The lack of this capability results in such problems as belated reaction to changes in requirements and inability to consider a reasonable number of alternatives.

## 2.0 A Corporate Memory Facility

To address these problems, we are studying the problem of building an intelligent, active corporate memory facility which would provide for the capture of the requirements and standards of a program, alternatives considered and trade-offs made over its lifetime, and relationships between these. The corporate memory facility would provide for requirements traceability, impact assessment, automation and/or assistance in the generation and evaluation of alternatives, and configuration management.

The corporate memory facility would support interactive problem solving across diverse areas such as the aerospace engineering disciplines (propulsion, weights, and aerodynamics). In operational use, a corporate memory facility would reduce life-cycle flow time and cost and improve the quality of program deliverables. Similar benefits could be realized by applying information accumulated in the corporate memory facility for one program to other related programs.

In initial phases of this work, the Advanced Technology Center is studying core corporate memory facility ideas, preparing corporate memory facility technical reports detailing study results, and building feasibility demonstrations. In conjunction with NASA, the Space Station Freedom Program was selected as a testbed; within this test bed we are concentrating on design knowledge capture. In 1989 the Advanced Technology Center examined aspects of the Power subsystem and the Environmental Control and Life Support (ECLS) subsystem. We also used our tools in a portion of the 1989 Space Station Freedom technical audit to investigate the rationale for a previous design decision.

Through the series of demonstrations, we are showing a novel integration and extension of design knowledge capture ideas by:

a. Tailoring knowledge acquisition and process control tools for *engineering trade studies*, a significant and feasible part of design knowledge capture.
b. Digitally recording speech as an unobtrusive method of capturing design rationale at the trade study workstation.
c. Developing an interactive design alternative generation aid.

## 3.0 Design Knowledge Capture

The Space Station Freedom was selected as the focus of research efforts toward a corporate memory facility since it is a large NASA project in a relatively early stage of design, and much of the design rationale could be captured or retrieved before it was lost. Many organizations in NASA and their subcontractors are interested in design knowledge capture, especially as it applies to the Space Station Freedom.

NASA's goal is to provide for a maximum of ease in the evolution of the Space Station Freedom and its adaptation to new requirements, new technologies, and advanced forms of machine intelligence. One facet of this is the Design Knowledge Capture Plan (NASA, 1988b). It is recognized that this goal must be pursued not only in the design of the Space Station Freedom, but also in the requirements for the documentation of the design, its features, and its rationale.

Following are objectives for design knowledge capture for the Space Station Freedom (Anon., 1988):

a. Establish design and development history for the Space Station Freedom Program.
b. Establish design and development traceability for the Space Station Freedom Program.
c. Maintain viable and effective risk management (e.g. failure modes and contingencies).
d. Capture and retain Space Station Freedom Program experience and expertise.

Design knowledge is defined as all physical descriptions of a system and its components; rationale for design decisions; functional flow diagrams; documentation of design objects or processes they perform along with the results; and interrelationships among design knowledge elements (such as part numbers and descriptions). Design knowledge embodies design objects and their attributes, including both designs selected for implementation and those not selected. It includes the rationale for requirements leading to design, methods of verification, exceptions and waivers from requirements, and other design criteria. Design knowledge is generated by engineers, management, technicians, and production teams (NASA, 1988b; Carnes, Olson, and Praharaj, 1988; Lakin et al., 1988; Anon., 1988; Sivard et al., 1989; NASA, undated).

Design knowledge capture encompasses the acquisition, storage, and manipulation of engineering data, information, and knowledge generated during a program (NASA, 1988b; NASA, undated). Design knowledge capture is also defined as the activity of transferring design knowledge from a source to a machine-interpretable form (NASA, 1988a; Wechsler and Crouse, undated).

Several sources have identified potential benefits of design knowledge capture (Purves and Carnes, 1987; NASA, 1988a; Beazley, 1988; Carnes, Olson, and Praharaj, 1988; anon. 1988; Sivard et al., 1989; Wechsler and Crouse, undated). Many of these benefits generalize to other portions of the corporate memory facility. They include:

a. Convergence of requirements and design through developing a rationale during the design process as each requirement is fulfilled.
b. Higher integrity of the design process and the attendant data products.
c. Better traceability of design data.
d. Integration of the Space Station Freedom Program across both logical and physical interfaces.
e. Secondary use of discipline data in the associated areas of failure modes and effects, resource management, integrated logistics support, configuration management and knowledge-based systems.
f. More efficient work efforts and products.
g. On line availability of design and decision data for critical systems.
h. Requirements traceability.
i. Design verification and validation.
j. Manufacturing quality control.
k. Documentation production.
l. Tutorials for the next generation of design engineers.
m. Effective management of engineering change and increased product quality through a common platform for engineering design, analysis, test, manufacturing.
n. An accumulated body of program knowledge that can fuel applications that manage life cycle functions beyond delivery.
o. Reduced sensitivity to personnel volatility.
p. Support for continuing engineering analysis.
q. Support for manufacturing.
r. Support for future applications.

There are also potential uses of captured design rationale in program management applications (Carnes, Olson, and Praharaj,1988). These include:

a. The influence of individual requirements can be traced.
b. The requirements source and reasoning process behind each design feature can be traced.
c. The influence of proposed requirements changes on the design can be assessed.
d. The effects of changes in assumptions can be assessed.
e. The overall reasoning process can be reviewed for possible future improvements.

## 4.0 Automating Engineering Trade Studies

We are focusing on trade studies in the design knowledge capture area because -

a. They exhibit a microcosmic path through the full cycle of design information, including requirements linkage, generation and comparison of alternatives, and decision documentation.
b. Many design engineers are familiar with trade studies and are comfortable using them to compare alternatives in quantitative terms.
c. Even though different methodologies for trade studies are available, little has been done to automate them.
d. A trade study tool would be immediately useful in a variety of domains, regardless of the success of the overall design knowledge capture or corporate memory facility effort.
e. Existing Advanced Technology Center tools could be extended to help perform portions of trade studies.

Trade studies are performed, in part, to avoid a designer's tendency to go directly to a design based on past experience, rather than trying to find a design that may better satisfy overall program requirements. Trade studies are often performed to help establish overall system configurations, study the detailed design of individual configuration items to provide the most cost-effective solution, and evaluate alternate solutions when the need for change occurs.

There are two general types of trade study criteria: limits which must be satisfied by any candidate system go/no go criteria (or *hard constraints*), and attributes upon which a ranking can be based (*soft constraints*).

Candidates are usually filtered using hard constraints and then ranked for comparison using soft constraints. Trade trees are used to decompose large numbers of candidates into groups for tractability. Paths through the tree show total configurations. Typical trade study criteria include accuracy, lifetime, power output, stability, sensitivity, bandwidth, low weight, low power, minimum dimensions, operational simplicity, electromagnetic compatibility, reliability, survivability, schedule, cost, safety, and risk. Criteria are usually weighted. The results are usually shown in a *trade study matrix* - a table showing the alternatives, criteria, ratings, and weights.

After candidates are rated and scored, a sensitivity analysis can be performed. This shows the sensitivity of the decision to changes in the value of attributes, weights, costs, and subjective estimates.

In our early work on the corporate memory facility we demonstrated the capture of trade study information and rationale (Figure 1). In the future, this information will be available through the Technical and Management Information System (TMIS). This system is available to anyone working on Space Station Freedom. We are examining several report formats based on current trade study practices and TMIS requirements. The information necessary for these reports is providing the foundation for the knowledge capture process.

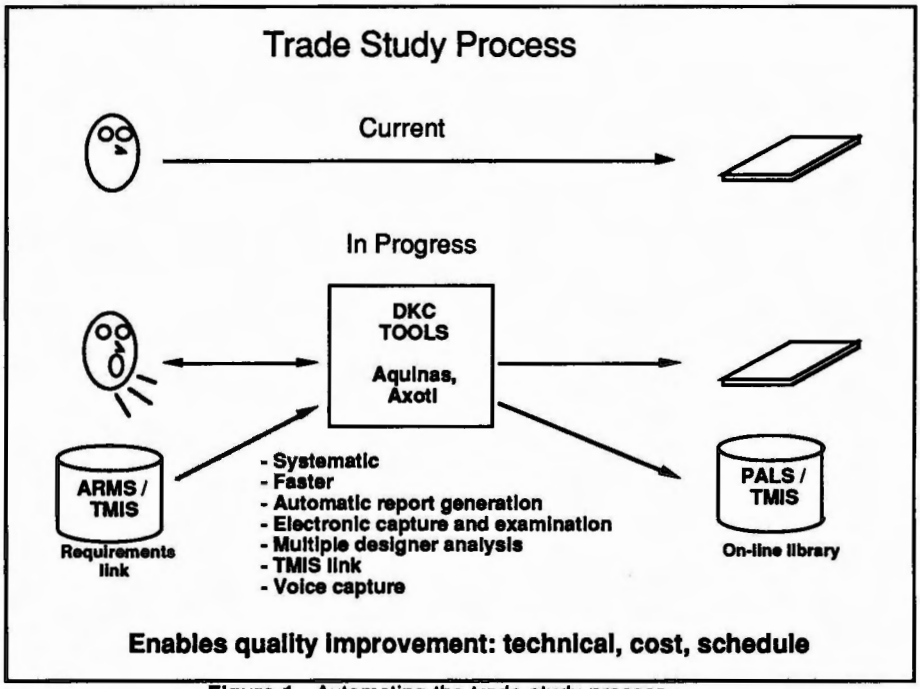



**Figure 1.** Automating the trade study process.

## 5.0 Design Knowledge Capture Tools

Two tools, Aquinas and Axotl, were used to build the first demonstration. An additional set of tools (MANIAC, HyperCard, and MacRecorder) was used to capture voice rationale and associate it with the Aquinas knowledge base for interactive playback.

### *5.1 Aquinas: Capturing Trade Study Design Rationale*

Aquinas interviewed experts in several trade study domains and captured candidate and criteria information leading to rank-ordered candidate selections. In the power domain, additional rationale was captured as voice input. In the ECLSS domain, conflicting opinions from multiple designers were captured, analyzed, and documented.

**Power subsystem** - Chuck Olson, a design engineer in Boeing Aerospace, used Aquinas to build two separate trade studies for the interface between a computer and automatic circuit breakers. Brian Smith, another Boeing Aerospace design engineer, offered advice on building an electronic trade study process assistant.

**Environmental Control and Life Support subsystem** - Jim Knox, a NASA design engineer at Marshall Space Flight Center, used Aquinas to build a trade study for carbon dioxide removal on Space Station Freedom in the year 2000. Allen Basckay, another NASA design engineer at Marshall Space Flight Center, added additional information to this trade study.

**Technical Audit Item #85** - John Palmer, O'Keefe Sullivan, and Carl Case, Boeing Aerospace, used Aquinas to document a 1986 decision about the placement of the pressurized logistics module.

Aquinas is a workbench developed by the Boeing Advanced Technology Center for acquiring and analyzing expert knowledge for solving diagnostic, structured selection, classification, and other problems (Figure 2). In the corporate memory facility context, Aquinas is used to acquire knowledge about requirements and alternatives from individuals or groups of experts, and then assists in merging that knowledge into a single knowledge base. Weights may be assigned to both requirements and their refinements. This knowledge may be merged automatically by Aquinas or by consensus of the program staff using Aquinas as an assistant. Aquinas supports similar capabilities for acquiring compound alternatives.

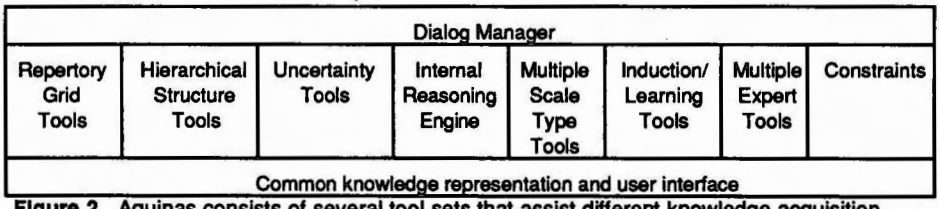



**Figure 2.** Aquinas consists of several tool sets that assist different knowledge acquisition tasks. General advantages of Aquinas include integration of multiple methods and techniques, rapid prototyping and feasibility analysis, generation of expert enthusiasm, multiple mediating representations, embedded testing, and life cycle support for verification, delivery, and maintenance.

Aquinas, an expanded version of the Expertise Transfer System (ETS; Boose, 1984, 1985, 1986a,b), combines ideas from psychology and knowledge-based systems to support knowledge acquisition tasks. These tasks include eliciting distinctions, decomposing problems, combining uncertain information, incremental testing, integration of data types, automatic expansion and refinement of the knowledge base, use of multiple sources of knowledge, use of constraints during inference, and providing process guidance (Boose and Bradshaw, 1987; Boose, Bradshaw, and Shema, 1989). Aquinas interviews experts and helps them analyze, test, and refine knowledge. Expertise from multiple experts or other knowledge sources can be represented and used separately or combined. Results from user consultations are derived from information propagated through hierarchies.

Using Aquinas, rapid prototypes of knowledge-based systems can be built in as little as one hour, even when the expert has little understanding of knowledge-based systems or has no prior training in the use of the tool. The interviewing methods in Aquinas are derived from George Kelly's Personal Construct Theory and related work (Kelly, 1955; Shaw and Gaines, 1987;

Boose, 1988). Kelly's methods and theory provide a rich framework for modeling the qualitative and quantitative distinctions inherent in an expert's problem-solving knowledge.

Aquinas tools mentioned here are explained more fully elsewhere (Boose and Bradshaw, 1987; Boose, 1988; Kitto and Boose, 1988; Shema and Boose, 1988; Bradshaw and Boose, 1990).

Extended repertory grids in Aquinas are a compact and easily understood form of expertise representation for many types of knowledge. Repertory grids can be analyzed, refined, tested, and maintained more easily than a corresponding, larger rule or frame knowledge base. In Aquinas, we have augmented repertory grid structures to include hierarchies, constraints, structures for eliciting and reasoning about knowledge from multiple experts, multiple variable types, and accommodate forms of machine learning. Generally, these analysis capabilities and compact, higher-level *mediating representations* of expert knowledge make knowledge bases easier to inspect, analyze, maintain, test, and improve. We use a test case-based approach within Aquinas for performance measurement, verification, and maintenance, and automatic knowledge base improvement. This method helps find holes and weaknesses in the knowledge base, and provides facilities for verifying knowledge consistency, accuracy, and sanity range.

Refinement methods in Aquinas include implication and similarity analyses, completeness checking, hole filling, cluster analyses, generalization, automatic rule production, internal testing and debugging aids, and graphic representation transformation. Expertise from multiple experts or other knowledge sources can be represented and used separately or combined, giving consensus and dissenting opinions among groups of experts. Recent progress on Aquinas has been in the areas of knowledge base performance measurement, knowledge base maintenance, interacting trait constraints, consultation graphics, and eliciting strategic and procedural knowledge. Experiments show how Aquinas can automatically improve knowledge bases and even suggest new problem-solving information. Forms of interactive and automatic machine learning are also employed by Aquinas (Boose, Bradshaw, and Shema, 1989).

Aquinas exists in several "C"-based versions that run on different microprocessor platforms and a fuller development version that runs on Sun workstations and Xerox Lisp Machines.

### *5.2 The Axotl System: Process Model Capture*

In the first demonstration, Chuck Olson used Axotl to elicit an electronically-based model of the trade study process.

Axotl, developed at the Boeing Advanced Technology Center, integrates a set of computer-based decision analysis tools with a knowledge-based system. The decision analysis tools are designed for problems requiring careful consideration of uncertainty and complex tradeoffs. In the context of corporate memory facility, alternatives and requirements generated by Aquinas can be analyzed using decision analysis representations to determine the suitability of various alternatives and to gauge the impact of changes in design requirements or circumstances. Influence diagrams are used to represent information, alternatives, and preferences both graphically and mathematically. Our experience has shown that they are an effective way of communicating important issues among participants. Axotl also employs other forms of knowledge representation that may prove useful as part of a corporate memory facility. For example, Boeing has extended and generalized an AND/OR graph representation for goals and activities ("activity graphs") that can be used to dynamically construct and evaluate cyclic plans for achieving a set of process requirements.

Axotl is written in the ParkPlace Smalltalk-80 development environment on the Apple Macintosh II. Versions of Smalltalk-80 exist for Sun, Apollo, Hewlett-Packard, IBM, and Apple hardware.

### *5.3 MANIAC, HyperCard, and MacRecorder: Capturing Voice Rationale*

Together, MANIAC, HyperCard, and MacRecorder were used to record and play back voice rationale.

In the first demonstration, design decision rationale was captured on a tape recorder during Aquinas sessions. To demonstrate feasibility, parts of these recordings were processed using MacRecorder on a Macintosh and stored in HyperCard. MANIAC, an Advanced Technology Center shell that controls communication between Axotl, Aquinas, HyperCard, and other application programs, receives commands from Aquinas to play back digitally recorded voice

based on particular Aquinas knowledge base objects. Designers and others who later examine the trade study decision rationale can optionally play back this recorded voice information.

In future demonstrations we will link MacRecorder and Aquinas more directly so that designers may enter and edit voice input directly while using Aquinas. This will be a relatively unobtrusive way to enter rationale (as opposed to text entry) in a cost effective manner. Digitally recorded voice information could eventually be stored and played back as design decision rationale in TMIS in a manner similar to many digital phone message systems.

MANIAC is described more fully in (Bradshaw, Covington, Russo, and Boose, 1990).

### *5.4 CANARD: Exploratory Design Alternative Generation*

As part of the design process, competing alternatives are generated and evaluated for suitability. The best alternative emerges as the result. Unfortunately, constraints, tradeoffs, and other considerations made during the exploration of the design are usually lost, making it impossible to review or easily modify them at a later time. If a modification to the design is required, the designers may have to redo the entire task.

We started development of CANARD, an automated tool which uses possibility tables, constraints, and knowledge bases to capture significant portions of the design process and assist in the generation of alternative solutions consistent with design goals and design constraints (Shema, Bradshaw, Covington, and Boose, 1990). Using a possibility table, a designer identifies the components of an acceptable design, specifies possibilities for each component, develops criteria reflecting preferences among possibilities, and supplies constraints governing compatibility between components and overall design considerations. The designer next interactively explores design alternatives by selecting possibilities for each component, modifying and/or adding components and possibilities as insight into the solution is gained. He then analyzes and stores the many alternative solutions for later retrieval.

For large problems, an iterative search procedure hypothesizes new constraints based on examples of previously-defined design alternatives, and proposes new design alternatives based on permutations of the constraint space. The tool keeps track of what has been tried and assists the designer in covering important aspects of the possible solution space.

CANARD is written in the ParkPlace Smalltalk-80 development environment on the Apple Macintosh II. Versions of Smalltalk-80 exist for Sun, Apollo, Hewlett-Packard, IBM, and Apple hardware.

## 6.0 Example Trade Study - Technical Audit Item #85

In 1989 a technical audit was performed on the Space Station Freedom for the program's content and implementation planning in relationship to performance, design, and validation requirements. One concern raised during the technical audit was a 1986 decision about the placement of the pressurized logistics module (PLM). Using Aquinas, we hoped to develop a process for capturing the decision rationale on this topic and similar ones.

First we described our problem and proposed process to a group of designers at Boeing in Huntsville, Alabama, who were or who are involved with the placement of the PLM. We then used Aquinas in two sessions with two teams of designers. One session lasted 1-1/4 hours, one session lasted 1-1/2 hours. We elicited trade study matrices from each team and combined the results, using Aquinas to show the combined rank-ordering. The decisions developed using Aquinas agreed with and documented the current placement of the PLM.

Here we describe the steps that were performed with Aquinas for the technical audit.

**Step 1.** Aquinas elicited nine alternative PLM locations from Team 1 (Node 1 Zenith, Node 1 Nadir, etc.).

**Step 2.** Aquinas elicited a preliminary set of decision criteria by using triadic comparison. Groups of three solutions were compared and designers were asked to give discriminating criteria:

> Think of an important new criterion that two of NODE.1.ZENITH, NODE.1.NADIR, and NODE .2.ZENITH share, but that the other one does not. What is that trait? (Enter a CR to skip over.)
> NEW TRAIT (EXTREME)** BETTER MSC REACH

What is that criterion's opposite as it applies in this case?
NEW TRAIT (OPPOSITE)** WORSE MSC REACH

What is the name of a scale or concept that describes BETTER.MSC.REACH / WORSE.MSC.REACH?
NEW TRAIT (CONCEPT)** MSC REACH

Think of an important new criterion that two of NODE.1.NADIR, NODE.2.ZENITH, and NODE.2.NADIR share, but that the other one does not. What is that characteristic? (Enter a CR to skip over.)
NEW TRAIT (EXTREME)** CLOSE TO HAB MODULE

What is that criterion's opposite as it applies in this case?
NEW TRAIT (OPPOSITE)** FARTHER FROM HAB MODULE

What is the name of a scale or concept that describes CLOSE.TO.HAB.MODULE / FARTHER.FROM.HAB.MODULE?
NEW TRAIT (CONCEPT)** HAB MODULE PROXIMITY

**Step 3.** The designers rated each alternative on each criterion. By default, Aquinas supplies ordinal scales from 1 to 5. Designers may change the scale type (to nominal, interval, or ratio) or range for convenience or more precision.

**Step 4.** The designers assigned a relative weight to each criterion. At this point an initial trade study matrix was complete (Figure 3).

TEAM.1. SOLUTION TRAIT

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 4 | 1 | 4 | 1 | 2 | 3 | 2 | 2 | 1 |
| 4 | 4 | 1 | 1 | 1 | 4 | 3 | 2 | 2 |
| 1 | 1 | 1 | 1 | 2 | 4 | 4 | 4 | 4 |
| 4 | 4 | 1 | 1 | 1 | 5 | 5 | 4 | 4 |
| 2 | 1 | 2 | 1 | 5 | 4 | 4 | 5 | 4 |
| 5 | 5 | 5 | 5 | 5 | 3 | 3 | 3 | 1 |
| 1 | 5 | 1 | 5 | 3 | 1 | 3 | 3 | 5 |
| *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* |

1. NODE.1.ZENITH
2. NODE.1.NADIR
3. NODE.2.ZENITH
4. NODE.2.NADIR
5. NODE.2.PORT
6. NODE.3.ZENITH
7. NODE.3.STARBOARD
8. NODE.4.PORT
9. NODE.4.NADIR

TEAM.1.SOLUTION

*1.* (5) MSC.REACH: BETTER.MSC.REACH(1)/ WORSE.MSC.REACH(5) [ORDINAL 1]
*2.* (1) HAB.MODULE.PROXIMITY: CLOSE.TO.HAB.MODULE(1) / FARTHER.FROM.HAB.M
*3.* (3) CG.SHIFT.IMPACT: LESS.CG.SHIFT(1) / GREATER.CG.SHIFT(5) [ORDINAL 1]
*4.* (1) JAPANESE.MODULE.PROXIMITY: CLOSER.TO.JAPANESE.MODULE(1) / FARTHER
*5.* (2) GROWTH./PATH: BETTER.FOR.GROWTH.PATH(1) / WORSE.FOR.GROWTH.PATH(
*6.* (5) PAYLOAD.BAY.BLOCKING: BLOCKING.PAYLOAD.BAY(1) / NOT.BLOCKING.PAYL
*7.* (4) EXPOSURE.TO.MICRO.METEOROIDS: MORE.EXPOSURE.TO.MICRMETEOROIDS(

**Figure 3.** Initial technical audit trade study matrix from Team 1.

**Step 5.** The designers used several of Aquinas' analysis tools to discover patterns in the collected information. Implication analysis showed logical generalizations that, for this application, provided a sanity check. A cluster analysis and similarity analysis showed the degree of similarity and redundancy between alternatives and between criteria.

**Step 6.** Aquinas scored the alternatives by eliciting preferred criteria values from the designers. For example, the designers said they would prefer alternatives that were *better for the station growth path* and had *less effect on the station center of gravity*. For Team 1, Aquinas produced the following results:

| | |
|---|---|
| 1 : NODE.2.NADIR | ( 1.00 ) |
| 2 : NODE.1.NADIR | ( 0.93 ) |
| 3 : NODE.2.PORT | ( 0.71 ) |
| 4 : NODE.2.ZENITH | ( 0.61 ) |
| 5 : NODE.4.NADIR | ( 0.54 ) |
| 6 : NODE.1.ZENITH | ( 0.54 ) |
| 7 : NODE.4.PORT | ( 0.48 ) |
| 8 : NODE.3.STARBOARD | ( 0.48 ) |
| 9 : NODE.3.ZENITH | ( 0.31 ) |

**Step 7.** The second team used Aquinas to independently develop and analyze their own trade study matrix.

**Step 8.** Both matrices were combined and Aquinas again scored the alternatives, this time showing the *consensus* scores as well as the contributions from both individual teams. The teams are weighted in this example for purposes of illustration (Team 1 has received a weight of 40%, Team 2 a weight of 60%). Teams or individuals may be weighted for technical or other reasons.

**Combined results:**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | NODE_2_NADIR 0.89 | TEAM_1 | 1.00 | 40% | TEAM_2 | 0.82 | 60% |
| 2 | NODE_1_NADIR 0.81 | TEAM_1 | 0.93 | 40% | TEAM_2 | 0.73 | 60% |
| 3 | NODE_2_PORT 0.71 | TEAM_1 | 0.71 | 40% | TEAM_2 | 0.70 | 60% |
| 4 | NODE_2_ZENITH 0.64 | TEAM_1 | 0.61 | 40% | TEAM_2 | 0.65 | 60% |
| 5 | NODE_1_ZENITH 0.55 | TEAM_1 | 0.54 | 40% | TEAM_2 | 0.56 | 60% |
| 6 | NODE_4_NADIR 0.48 | TEAM_1 | 0.54 | 40% | TEAM_2 | 0.44 | 60% |
| 7 | NODE_4_PORT 0.43 | TEAM_1 | 0.48 | 40% | TEAM_2 | 0.39 | 60% |
| 8 | NODE_3_ZENITH 0.40 | TEAM_1 | 0.31 | 40% | TEAM_2 | 0.46 | 60% |
| 9 | NODE_3_STARBOARD 0.28 | TEAM_1 | 0.48 | 40% | TEAM_2 | 0.15 | 60% |

Given this information, Aquinas displayed the most dissenting opinion beside the consensus. The dissenting opinion is found by computing a correlation score between each team and the consensus; the team with the lowest correlation score is listed as the dissenting opinion. Dissenting opinions show the user the range of opinion about a decision, not just the top rated list. In this case, both teams showed a high correlation - both teams were in substantial agreement. This can give the user confidence that the top rated alternatives were sound choices.

**Correlation scores for all experts:**

TEAM_2 .96
TEAM_1 .90

TEAM_1 has the most dissenting opinion.

| **TEAM_1** | / | **Consensus** |
|---|---|---|
| 1 : NODE_2_NADIR 0.81 | / | NODE_2_NADIR 0.89 |
| 2 : NODE_1_NADIR 0.72 | / | NODE_1_NADIR 0.81 |
| 3 : NODE_2_PORT 0.71 | / | NODE_2_PORT 0.71 |
| 4 : NODE_2_ZENITH 0.66 | / | NODE_2_ZENITH 0.64 |
| 5 : NODE_4_NADIR 0.62 | / | NODE_1_ZENITH 0.55 |
| 6 : NODE_1_ZENITH 0.42 | / | NODE_4_NADIR 0.48 |
| 7 : NODE_3_STARBOARD 0.41 | / | NODE_4_PORT 0.43 |
| 8 : NODE_4_PORT 0.34 | / | NODE_3_ZENITH 0.40 |
| 9 : NODE_3_ZENITH 0.21 | / | NODE_3_STARBOARD 0.28 |

**Voice capture.** Decision rationale was captured as voice input for both teams. Decision rationale included descriptions of the overall problem, rationale for narrowing the alternatives to those appearing in the matrix, assumptions underlying decisions, definitions of alternatives and criteria, reasons for assigning particular ratings and criteria weights, and realistic assumptions concerning constraints and tradeoffs. Of particular interest were situations where members of the same team initially disagreed and eventually reached consensus. These discussions, when played back, are particularly illuminating for decision makers and others who may need to update trade studies when requirements change.

**Near-future capability.** We will be building a TMIS-based menu query mechanism that would be able to answer several types of questions about a trade study:

**Q.** Why did NODE.2.ZENITH do better than NODE.1.ZENITH?
**A.** It rated higher on CLOSE.TO.HAB.MODULE (1 vs. 4 on a scale of 1 to 5) and CLOSE.TO.LAB. MODULE (1 vs. 4 on a scale of 1 to 5)

**Q. Why did NODE.1.NADIR and NODE.2.NADIR do better than NODE.1.ZENITH and NODE.2.ZENITH?**
**A. They always rated higher on BETTER.MSC.REACH, BETTER.FOR.GROWTH, and LESS.EXPOSURE.TO.MICROMETEOROIDS.**
**They sometimes rated higher on CLOSE.TO.HAB.MODULE and CLOSER.TO.JAPANESE.MODULE.**

**Q. If LESS.EXPOSURE.TO.MICROMETEOROIDS were the only criterion, how would the alternatives be ranked?**
**A. 5: NODE.1.NADIR, NODE.2.NADIR, NODE.4.NADIR**
**4: -**
**3: NODE.3.STARBOARD, NODE.4.PORT**
**2: NODE.2.PORT**
**1: NODE.1.ZENITH, NODE.2.ZENITH, NODE.3.ZENITH**

**Q. What criteria most discriminate between the alternatives?**
**A. GROWTH.PATH, MSC.REACH (TEAM_1), SUPPLY.ROUTE , TRAFFIC.PATTERN (TEAM_2)**

# 7.0 Discussion

## *7.1 Project Benefits*

NASA concluded that "the Aquinas-based methodology was appropriate for effecting most of the rationale capture required for the Space Station Freedom Program, and was especially well suited to trade studies" (Freeman, 1989). They felt that "the captured rationale will be an invaluable resource for the Space Station Freedom Program over its entire lifetime by providing immediate, reliable access to how and why important decisions were made." Benefits of rationale capture were listed as:

a. Better, more reliable design
   - More comprehensive identification of alternatives
   - Explicit rationale can be analyzed, discussed, and agreed upon
   - Design is less "scenario dependent"
b. Reduced development cost
   - Design review will be greatly enhanced, resulting in fewer costly and risky "fixes" late in the program
   - Reduced reliance on inefficient and less effective paper-based methods
   - Less need to pull people in for a meeting to reach an understanding of design elements
c. Reduced cost for operations and maintenance
   - Immediate, full access is provided to the designer's rationale in considering repairs or procedures which deviate from "standard"
d. Reduced cost for follow-on design efforts of the Space Station Freedom Program
   - Evolutionary design and augmentations do not have to "reinvent the wheel" by reconstructing the original design rationale, even if the designers have been unavailable for ten, twenty, or thirty years.
e. Reduced cost for design of similar artifacts
   - Mars mission, lunar colony

Based on this work, NASA is sponsoring the development of a Design Alternatives Rationale Tool (DART) for use on the Space Station Freedom Program. It will be based on extended repertory grids and other methods and will be specialized design for rationale capture. NASA plans to use DART during the preliminary design review for the Space Station Freedom. Like Aquinas, DART could also find use as a general decision aid, a group decision tool, a feasibility assessment aid, and as a knowledge engineering tool. Eventually, NASA would like to use captured rationale to support knowledge-based systems including an Intelligent Design Assistant for Evolutionary Design (Freeman, 1989).

We are continuing to study and model the trade study process. This should lead to further improvements in quality and efficiency. In particular we will (1) recommend a process for making voice playback available over TMIS, (2) help NASA establish guidelines and standards for criteria

weights and alternative ratings, and (3) help NASA develop standard criteria templates and checklists to make trade studies more complete and consistent across the Space Station Freedom Program.

### *7.2 Summary*

The Boeing Advanced Technology Center is conducting research leading to a corporate memory facility. A corporate memory facility would provide facilities for capturing and using decision history and rationale throughout a major program's life cycle.

Initially the Advanced Technology Center is preparing corporate memory facility technical reports and building feasibility demonstrations. In conjunction with NASA, The Space Station Freedom Program was selected as an application; within this domain we are concentrating on design knowledge capture. We examined aspects of the Power subsystem and the Environmental Control and Life Support (ECLS) subsystem. We also participated in one aspect of the Space Station Freedom technical audit.

Significant progress was made in helping automate the process of performing engineering trade studies. Other steps in the design knowledge cycle - alternative generation, comparison, evaluation, and documentation - were also demonstrated. In the next phase we will continue to extend our tools to further automate trade studies, strengthen our links to TMIS, and continue work on CANARD.

## Acknowledgements

Thanks to Guy Boy, Kathleen Bradshaw, Ray Carnes, Carl Case, Arch Cousins, Stan Covington, Walter Cowart, Tom Dollman, Bob Drosdzal, Larry Frank, Mike Freeman, Peter Friedland, Scott Herman, Randy Humphries, Randy Johnson, Rich Keller, Cathy Kitto, Jim Knox, Wendy Nikolich, Guy Olney, Chuck Olson, John Palmer, Tim Rau, Tim Saito, Cecelia Savaard, Suzzanne Shema, O'Keefe Sullivan, Gregg Swietek, Bryan Walls, David Weeks, and Al Winters for their advice, contributions, and support. ETS, AQUINAS, AXOTL, CANARD, and MANIAC were developed at the Boeing Advanced Technology Center, Boeing Computer Services, Seattle, Washington.

## References

Anon. (1988). Design Knowledge Capture, NASA viewfoil presentation, Feb. 5.

Beazley, W. G. (1988). Design Knowledge Capture, A Methodology and Strategy, Final Report for NASA, Grumann/PSC, Task Order SE 8805-WGBA by William G. Beazley and Assoc., Texas.

Boeing Computer Services (1989a). Information to be Retained in a Corporate Memory Facility, CMF TR1, NASA NAS2-121108, April.

Boeing Computer Services (1989b). Corporate Memory Facility Demonstration 1: Knowledge Representation and Trade Studies, CMF DEMO1, NASA NAS2-121108, April.

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., 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, 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., Bradshaw, J. M., Shema, D. B. (1989). Recent Progress in Aquinas: A Knowledge Acquisition Workbench, *Knowledge Acquisition: An International Journal*, Vol. 1, No. 2, pp. 185-214.

Bradshaw, J. M., Covington, S. P., Russo, P. J., and Boose, J. H. (1990). Folie a Deux: Integrating Aquinas, a Personal-Construct-Based Knowledge Acquisition Workbench, with Axotl, a Knowledge-Based Decision Analysis Workbench, *International Journal of Man-Machine Studies*, in press.

Bradshaw, J. M., and Boose, J. H. (1990), Decision Analysis Techniques for Knowledge Acquisition: Combining Information and Preferences Using AQUINAS, Special Issue on the Second Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, in press.

Carnes, J. R., Olson, A. E., and Praharaj, M. (1988). Design Knowledge Workstation, 1988 Final Report, Boeing Computer Services, PO Box 1470, M/S JA-74, Huntsville, AL.

Freeman, M. S. (1989). Space Station Freedom Program Technical Audit Action 85, presentation at NASA Level 2, Reston, September 20.

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

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

Lakin, F., Wambaugh, J., Leifer, L., Cannon, D., Sivard, C. (1988). The Electronic Design Notebook: Performing Medium and Processing Medium, NASA AMES Research Center, CA, December.

NASA (1988a). Process Requirements Document for Design Knowledge Capture, Space Station Program, March 28, NASA Parkridge Space Center, Reston, VA.

NASA (1988b). Space Station Program, Design Knowledge Capture Plan, Work Package One, Draft 1, D683-10267-3, December 22.

NASA (undated). Space Station Work Package 1, RFP No. 1-7-KA-00052-52G, George C. Marshall Space Flight Center, Alabama.

Purves, R. B. and Carnes, J. R. (1987). Design Knowledge Capture: Preserving Engineering Knowledge for Future Applications, Boeing Aerospace Company, PO Box 1470, MS JA-68, Huntsville, AL.

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.

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*, v. 29 n. 4, p.447-460.

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

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

Wechsler, D. B. and Crouse, K. R.(undated). An Approach to Design Knowledge Capture for the Space Station, The MITRE Corporation, 1120 NASA Road One, Houston, TX.

# Knowledge Acquisition via Knowledge Integration

**Pavel B. Brazdil**
**Luís Torgo**

**LIACC**
**Laboratory of AI and Computer Science**

University of Porto
Rua Campo Alegre, 823 - 2°
4100 Porto, Portugal
ltorgo, pbrazdil@liacc.up.ctt.pt

**Abstract.** In this paper we are concerned with the problem of *acquiring knowledge* by *integration*. Our aim is to construct an integrated knowledge base from several separate sources. The need to merge knowledge bases can arise, for example, when knowledge bases are acquired independently from interactions with several domain experts. As opinions of different domain experts may differ, the knowledge bases constructed in this way will normally differ, too. A similar problem can arise also whenever separate knowledge bases are generated by learning algorithms. The objective of *integration* is to construct *one* system that exploits *all* the knowledge that is available and has a good performance. The aim of this paper is to discuss the methodology of knowledge integration, describe the implemented system (INTEG.3) and present some concrete results which demonstrate the advantages of this method.

## 1. Introduction

The areas of *knowledge acquisition* (KA) and *machine learning* (ML) have until recently existed as two rather distinct areas, despite the fact that both areas share quite similar concerns. Both areas are attempting to construct systems that *model* certain phenomena in the real world so as to *aid us* in decision making.

Most of the research in KA pressuposes that the model is created by a person (an 'expert') first. The objective of *expertise transfer* (ET) tools is to help the user to transfer this expertise into a computer knowledge base first. In *machine learning*, on the other hand, the system is usually supplied with *data* which is stored in the knowledge base. The process of *model creation* is left to the system, that is, to the appropriate empirical inductive tool.

As Gains [1989] has pointed out each approach has its own difficulties. Experts do often provide a mixture of relevant and irrelevant or erroneous information. The objective of current expertise transfer tools, such as Aquinas, KSS0 or KADS [Wielinga & Breuker, 1986] is to help the user to clean up the information provided. The disadvantages of a pure machine learning approach are also quite obvious. The system is too dependant on the data that is provided from outside. This data need not necessarily be complete. Some important cases may simply be missing. Also, a part of the data may be corrupted. The data may contain incorrect information.

Some people have expressed the view that both methodologies should be combined [Gaines, 1989; Boose et al., 1989], but not many concrete proposals were put forward so far. Aquinas, for example, was capable of presenting to the user opinions of multiple experts, but the user was responsible for weighing opinions and selecting the solutions to be followed. Also, as Aquinas used *repertory grids* for representation of concepts, the language used for representing knowledge is somewhat limited. As Boose et al. [1989] point out further facilities are needed for elicitation and analysis of knowledge from multiple experts, resolving potential conflicts (e.g. by negotiation) and knowledge base merging.

In this paper we describe a method for merging several separate theories (knowledge bases)[1]. We assume that, in general, these will have been generated in different ways. That is, some theories can be obtained by querying an expert and transcribing his knowledge in the form of rules. Others can be obtained on the basis of data, using inductive learning tools (see Fig.1). System INTEG.3 is capable of analysing the individual theories and its rules, and selecting (or marking) some to be included in the integrated theory. These selected rules determine the decision of the overall system.

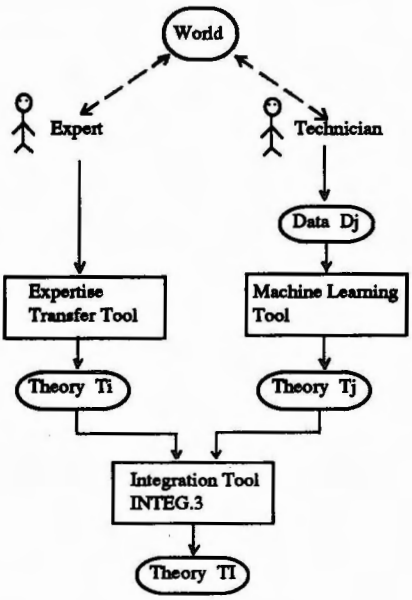



Fig. 1 Possible Use of Integration Tools

[1] In this paper the term "theory" is used interchangeably with the term "knowledge base". Both terms have a rather restricted meaning here. They are used to identify a collection of rules (e.g. Horn clauses).

Most of the experiments we have performed were concerned with a medical domain[2]. All the theories that were used by INTEG.3 were generated by inductive learning tools. The theories differed from one another in various ways, and it was not a priori clear which theory was right. In the experiments the integrated theory had significantly better performance than the original theories. The experiments confirmed our belief that knowledge integration can play an important role in the process of knowledge acquisition.

### Knowledge Integration and Incremental Learning

Knowledge integration is concerned with issues that are related to those in *incremental learning* systems, such as ID4 [Schlimmer and Fisher, 1988], ID5 [Utgoff, 1988], AQ16 [Janikow, 1989], for example. Both knowledge integration and incremental learning attempt to construct a theory that explains best the given data. There are some important differences between the two approaches, however.

When we talk about incremental learning, usually it is assumed that only *one system* is constructing theories by employing some incremental version of a given learning algorithm. In consequence all the data is actually analyzed by this system at one time or another. Knowledge integration, on the other hand, involves *several systems* all of which try to construct their own theories on the basis of their *own* experience. Although knowledge integration may require some common experience in the process of constructing an integrated theory, we are not in favour of simply transferring all the data to one system. People do not use only "raw data" when they communicate with others about certain world phenomena.

If we let the systems communicate theories, instead of data, some learning effort will be saved, but we can expect that some information will be lost in the process. We could thus expect that the results obtained by knowledge integration may not supercede the results obtained by incremental learning systems. Our experiments have indicated the contrary. The performance of the integrated theory was better than the performance of the incremental system (see Section 2.4 for more details).

Knowledge integration has, however, one additional advantage over incremental systems which are unable to improve upon theory provided by the user. Incremental learning systems keep various internal data structures and statistical measures in the memory (e.g. informativity of various attributes etc.). These enable the system to update its theory when new data gets available. If these are not supplied by the user (or alternatively, if the user does not provide all the examples he has seen), the incremental systems can do little to improve the given theory.

## 2. Method of Knowledge Integration

In this section we will describe the method of knowledge integration in more detail. Basically, the process involves a preparatory phase in which a group of systems (agents) is selected and organized into a group that can function together. The group can include systems $S_1$-$S_n$ that are capable of constructing theories from data, responding to external queries and communicating with one another. Here we shall assume that this has already been done and that the system organization is fixed [3].

Having defined the organization of the multi-agent system, it is necessary to determine the overall

---

[2] The tests referred to here were performed on lymphography data (obtained from JSI, Ljubljana) which contained 4 possible classifications. This may not be exactly the same data as the one used by Gams [1989] which contained 9 possible classifications.

[3] In general systems can be given the ability to reorganize themselves. These are some of the concerns of Distributed Artificial Intelligence (see e.g. [Bond and Gasser, 1988]), but they are outside the scope of this paper.

objectives. That is, for example, one can define which concepts are to be acquired and/or describe in some way the required performance. This is important if we want the multi-agent system to decide when to stop altering its theories. Here we will follow what has been done in the past and let the user control this issue.

Consequently, here we will be concerned with, basically, the following three phases:

(1) Generation of independent theories (by consultation or inductive learning),
(2) Competitive characterisation of system's theories,
(3) Construction of the integrated theory.

In phase (1) the systems $S_1$-$S_n$ work in an independent manner, and as a result produce theories $T_1$-$T_n$. Each system involved constructs its *own theory* on the basis of its own experience. Here $S_j$ can represent either a human, or an inductive learning tool. In either case $S_j$ will produce theory $T_j$.

In phase (2) the individual theories are *characterized* using tests. Without loss of generality let us assume that this is actually controlled by some agent SI. This agent poses a query to all the agents involved, waits for the answers and then proceeds with the next query. Any of the systems $S_1$-$S_n$ could act as SI. The subsystem responsible for characterization of theories is referred to as INTEG.3.

Phase (3) is dedicated to the issue of constructing one integrated theory (TI) on the bases of the results obtained in phase (2). This task is also done by INTEG.3. The three phases mentioned could be followed by two additional ones:

(4) Adoption of integrated theory by one (or more) systems,
(5) Check whether the process should continue, and if so, go to (1).

In this paper we will be concerned mainly with the phases (1)-(3). The issue of how one could construct a 'closed loop system', capable of taking the integrated theory and using it input in further learning will be discussed in a future paper. The next section describes phases (1)-(3) in more detail.

## 2.1 Generation of Individual Theories

### Use of Inductive Learning Tools

The decision concerning which system we should choose to generate individual theories is not really too important in the context of we want to prove. Here we require only that the system(s) are capable of generating theories that perform reasonably well on tests. As we had earlier reimplemented ID3- and AQ-like systems, we decided to use these as the basic inductive engines in our set-up.

The reimplementation of ID3 based on earlier work (e.g. Quinlan [1986], Cestnik et al. [1987], Clark and Niblett [1987; 1989]) will be referred to as ITL1 (*Inductive Tree Learning System*). The decision tree generated by this system is automatically converted into a rule form which we find more amenable for further manipulation.

The inductive rule learning system IRL1 is an incremental learning program, that was partially inspired by CN2 [Clark and Niblett, 1989]. This system incrementaly updates the existing rules either using generalization and/or specialization. More details about this system will be made available in [Torgo, 1990].

Different theories needed by the knowledge integration system (INTEG.3) were generated by the inductive learning systems (ITL1 and IRL1) in a series of *independent learning tasks*. In each task the inductive learning system generated a theory (consisting of a set of rules) on the basis of its own data. Let us see how the data was prepared for each experiment.

From the total number of cases available (D), 30% were separated out by a random process and

objectives. That is, for example, one can define which concepts are to be acquired and/or describe in some way the required performance. This is important if we want the multi-agent system to decide when to stop altering its theories. Here we will follow what has been done in the past and let the user control this issue.

Consequently, here we will be concerned with, basically, the following three phases:

(1) Generation of independent theories (by consultation or inductive learning),
(2) Competitive characterisation of system's theories,
(3) Construction of the integrated theory.

In phase (1) the systems $S_1$-$S_n$ work in an independent manner, and as a result produce theories $T_1$-$T_n$. Each system involved constructs its *own theory* on the basis of its own experience. Here $S_i$ can represent either a human, or an inductive learning tool. In either case $S_i$ will produce theory $T_i$.

In phase (2) the individual theories are *characterized* using tests. Without loss of generality let us assume that this is actually controlled by some agent $S_t$. This agent poses a query to all the agents involved, waits for the answers and then proceeds with the next query. Any of the systems $S_1$-$S_n$ could act as $S_t$. The subsystem responsible for characterization of theories is referred to as INTEG.3.

Phase (3) is dedicated to the issue of constructing one integrated theory (TI) on the bases of the results obtained in phase (2). This task is also done by INTEG.3. The three phases mentioned could be followed by two additional ones:

(4) Adoption of integrated theory by one (or more) systems,
(5) Check whether the process should continue, and if so, go to (1).

In this paper we will be concerned mainly with the phases (1)-(3). The issue of how one could construct a 'closed loop system', capable of taking the integrated theory and using it input in further learning will be discussed in a future paper. The next section describes the phases (1)-(3) in more detail.

## 2.1 Generation of Individual Theories

### Use of Inductive Learning Tools

The decision concerning which system we should choose to generate individual theories is not really too important in the context of we want to prove. Here we require only that the system(s) are capable of generating theories that perform reasonably well on tests. As we had earlier reimplemented ID3- and AQ-like systems, we decided to use these as the basic inductive engines in our set-up.

The reimplementation of ID3 based on earlier work (e.g. Quinlan [1986], Cestnik et al. [1987], Clark and Niblett [1987; 1989]) will be referred to as ITL1 (*Inductive Tree Learning System*). The decision tree generated by this system is automatically converted into a rule form which we find more amenable for further manipulation.

The inductive rule learning system IRL1 is an incremental learning program, that was partially inspired by CN2 [Clark and Niblett, 1989]. This system incrementaly updates the existing rules either using generalization and/or specialization. More details about this system will be made available in [Torgo, 1990].

Different theories needed by the knowledge integration system (INTEG.3) were generated by the inductive learning systems (ITL1 and IRL1) in a series of *independent learning tasks*. In each task the inductive learning system generated a theory (consisting of a set of rules) on the basis of its own data. Let us see how the data was prepared for each experiment.

From the total number of cases available (D), 30% were separated out by a random process and

reserved for the final tests of the integrated theory. Let us call this test set set DT. If we disregard the cases in this set, 70% of the original data can be available for the creation of alternative theories (see Fig. 2). These are constructed with the help of different inductive learning systems. In most of our experiments we have generated four theories. Some were generated by ITL1 and others by IRL1.

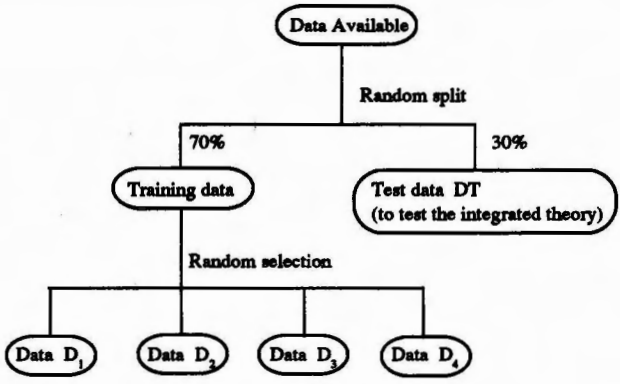



Fig. 2 Generation of Various Training Sets

In each experiment some cases were selected at random from the pool of data available (D-DT). Let us call this set $D_i$. Set $D_i$ was then supplied to the inductive learning system to generate theory $T_i$. This process was then repeated to generate the next theory (see Fig.3). As sets $D_1$ .. $D_n$ were selected at random from the same pool of examples, the sets mentioned could have some cases in common. We had no objection to that, because in real life people may also encounter cases identical to those used by others.

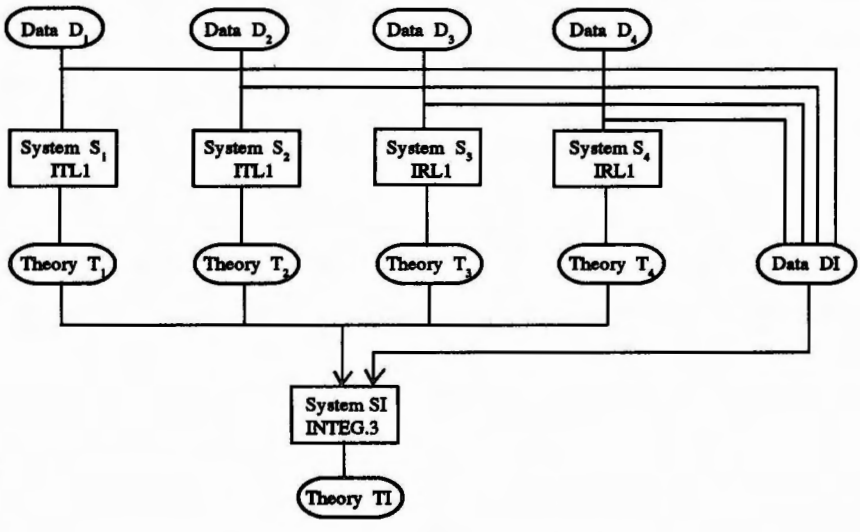



Fig. 3 The Set-up for Knowledge Integration Experiments

## 2.2 Competitive Characterization of Theories

After the theories have been constructed by the individual systems they are characterized on the basis of integration tests for which we need some data. Let us refer to this data as the set DI. The easiest way to obtain this set is to take the union of the data sets $D_1$.. $D_n$. This is the approach we have actually adopted. In general, however, the set DI may consist of any other representative sample of data.

The results of tests are both qualitative and quantitative. The objective of *qualitative* characterisation is to provide detailed information about relative benefits of individual rules or theories to the integration system. This information takes the form of lists of cases covered by a particular theory or rule. The objective of *quantitative* characterization is to estimate the overall *accuracy* of the individual theory (or rule).

### Qualitative Characterization of Theories

Qualitative characterisation of *theories* is similar to qualitative characterisation of *rules*, and so in the following we shall simply speak about qualitative characterization of rules.

Qualitative characterisation of a particular rule R consists of two lists. The first one mentions all the test cases that were *correctly* covered by this rule. In other words this list refers to the *positive examples* covered the rule. The second list mentiones all the cases that were *incorrectly* covered by the rule. This list represents the *negative examples* covered by the rule in question. The lists mentioned do not need to contain complete descriptions of each case. For our purposes it is sufficient to store only case identifiers (indeces) that uniquely identify each case.

### Quantitative Characterization of Theories

Quantitative characterization of a given theory is done using an estimate of accuracy. These are calculated on the basis of test results on the test set DI.

The tests are done in a usual manner by comparing the classification predicted by theory $T_i$ with the classification stored together with the case. This comparison enables us to decide whether the particular theory classified the case correctly or not. The classification errors caused by *misclassification* are sometimes referred to as errors of *commission*. Errors of *omission* arise whenever an expert or a system fails to classify some case, that is, when no classification is actually predicted.

The accuracy of theory *T* is estimated using the following formula:

$$AC_T = C_T / (C_T + EN_T + ER_T)$$
$$= 1 - (EN_T + ER_T) / (C_T + EN_T + ER_T)$$

where $C_T$ represents the number of correctly classified cases, $ER_T$ the number of misclassifications, and $EN_T$ errors of ommision. In effect the coefficient Qi represents a ratio of correctly classified cases ($C_T$) to all cases used in testing ($C_T + EN_T + ER_T$).

### Quantitative Characterization of Rules

Quality of individual rules is estimated in a heuristic manner, using the expression

$$Q_{T,R} = AC_{T,R} * \text{Estimate_of_coverage}$$

where $AC_{T,R}$ represents the *accuracy* of rule *R* and *Estimate_of_coverage* is a number between 0 and

1 that guides the system to prefer rules with relatively high coverage of a class. Its value is calculated as follows:

$$\text{Estimate_of_coverage} = \exp(C_{T,R} / N_{C,R} - 1)$$

The value $C_{T,R}$ represents the number of correct classifications. The value of $N_{C,R}$ is obtained as follows. The consequent of rule *R* determines which class (C) the rule is concerned with. Then $N_{C,R}$ represents the total number of examples of that class (C). The accuracy of individual rules is estimated in a similar way as accuracy of theories using:

$$AC_{T,R} = C_{T,R} / (ER_{T,R} + C_{T,R})$$

Here $AC_{T,R}$ represents the accuracy of rule *R* belonging to theory *T* and $ER_{T,R}$ represents the number of errors. We notice that this formula takes into account errors of commission but not errors of omission. The reason is that whenever several rules are associated with a particular class, and no rule is activated, it is difficult decide which rule is at fault. This is why we have decided not to include these errors in our estimates.

Let us consider an example. In our system, the assertion,

rule(1,5, metastase <— earlyuptakein = no $\wedge$ defectinnode = lac_central, 15, 3).

for example, represents rule 5 of theory $T_1$. The numbers 15 and 3 represent number of correctly classified cases ($C_{1,5}$) and number of misclassifications ($ER_{1,5}$). The accuracy of this rule is thus

$$AC_{1,5} = C_{1,5} / (ER_{1,5} + C_{1,5}) = 15 / (3 + 15) = 83\%$$

Supposing we have 30 examples of metastase then $Q_{1,5}$ would be calculated using:

$$Q_{1,5} = 0.83 * \exp(15/30 - 1) = 0.83 * 0.61$$

Both qualitative and quantitative characterizations of theories are used in the construction of the integrated theory.

## 2.3 Generation of the Integrated Theory

The integrated theory TI is constructed on the basis of the candidate set TC. Initially set TC contains all the rules of the individual theories $T_1..T_n$. The objective is to select some rules from TC and transfer them into TI so as to achieve good performance (accuracy). The method relies on the qualitative and quantitative characterization of rules and includes the following steps:

(1) Order rules in the candidate set according to their estimates of quality.
(2) Select the rule R with the best quality and include it in TI.
(3) Mark the cases covered by R.
(5) Recalculate the quality estimates of rules excluding the marked cases.
(6) Go back to (1).

The process of adding new rules to TI terminates when the accuracy of 'best rule' in TC falls below a certain threshold. In our experiments the limit was set to rather low value (1%). Even this low value, however, prevented the inclusion of many rules in the integrated theory. These were the rules that covered no examples (all the examples got already covered).

## 2.4. Results

We have conducted a series of experiments whose purpose was to compare the performance of the integrated theory with the performance of the original theories. The systems were asked to construct theories on the basis of 5, 10 .. 30 training examples. Each theory was obtained by an inductive process on the basis of a given number of examples which were drawn from a given pool by a random process. In order to exclude the possibility of fortuitous results each experiment was repeated ten times. The perfomance of system $S_1$, for example, obtained on the basis of 15 training examples was assessed by calculating the mean performance of 10 theories generated by this system. The mean performance in this case was 57.3% ( the standard deviation was 10.4%). The results of our experiments are shown in the following table.

| | 5 | 10 | 15 | 20 | 25 | 30 examples |
|---|---|---|---|---|---|---|
| $S_1$ | 39.3 (17.2) | 51.6 (14.2) | 57.3 (10.4) | 47.5 (9.4) | 58.2 (6.7) | 60.9 (7.8) |
| $S_2$ | 46.8 (16.5) | 55.4 (10.1) | 55.5 (14.9) | 59.1 (7.4) | 62.3 (5.5) | 56.8 (8.0) |
| $S_3$ | 52.9 (17.5) | 54.8 (15.9) | 55.2 (7.1) | 72.9 (10.3) | 57.1 (11.5) | 68.1 (3.8) |
| $S_4$ | 54.5 (16.1) | 57.0 (17.0) | 67.5 (10.8) | 66.6 (8.3) | 73.0 (10.4) | 75.7 (4.8) |
| $S_M$ | 48.4 (9.6) | 54.7 (9.1) | 58.9 (5.2) | 61.5 (4.7) | 65.1 (4.1) | 65.4 (3.2) |
| TI | 70.4 (12.9) | 73.6 (10.0) | 80.0 (7.5) | 82.7 (6.6) | 83.9 (6.3) | 85.4 (3.9) |
| Gain | 22.1 (8.8) | 18.9 (8.0) | 21.1 (5.1) | 21.2 (5.9) | 18.8 (6.9) | 20.1 (4.7) |

Table 1. Performance of the Integrated Theory TI and the Individual Systems ($S_1$-$S_4$).

The first four rows of the table show the performance figures of individual systems ($S_1$-$S_4$). Systems $S_1$,$S_2$ use ITL1 inductive learning method while systems $S_3$,$S_4$ use IRL1. All the figures shown are *means* obtained on the basis of ten independent experiments. The standard deviation is shown in brackets. The values of $S_M$ represent the mean value calculated as follows: $S_M = (S_1 + S_2 + S_3 + S_4) / 4$. This value is useful when trying to assess the performance gain of the integrated theory TI. The last row shows the gain (TI - $S_M$). As can be seen the performance gains were around 20%.

Fig. 4 shows some of the results in a graphical form. As we can see system $S_1$ follows a learning curve. It is interesting to note that the performance tends to fluctuate less as more examples are given. That is, the standard deviation decreases with the number of examples. The performance of the integrated theory exceeds the performance of $S_1$ (and the systems). The performance gains obtained were really quite surprising. They were around 20%. The integrated theory was also quite "stable" in the sense that its performance did not fluctuate as much as the performance of the individual systems.

An interesting question is whether one particular system, say $S_1$, could obtain a similar performance as the integrated theory (TI), had it been supplied with all the data. Here we shall call this system $S_1$+. We have conducted a separate experiment to compare the performance. The results are shown in Table 2. We have found it rather surprising that the integrated theory had better performance than system $S_1$+.

| | 5 | 10 | 15 | 20 | 25 | 30 examples |
|---|---|---|---|---|---|---|
| $S_1$+ | 50.0 (11.9) | 58.2 (11.4) | 63.7 (9.1) | 66.6 (9.1) | 66.8 (10.3) | 73.4 (7.4) |
| TI | 70.4 (12.9) | 73.6 (10.0) | 80.0 (7.5) | 82.7 (6.6) | 83.9 (6.3) | 85.4 (3.9) |
| Gain | 19.6 | 15.4 | 16.3 | 16.1 | 17.1 | 12.0 |

Table 2. Performance of the Integrated Theory and System $S_1$+.

The first row of this table shows the performance of system $S_1$ that has been additionally supplied with the data of systems $S_2$ - $S_4$. The extra data helped $S_1$ to improve its performance, but the performance of the integrated theory (TI) was better.

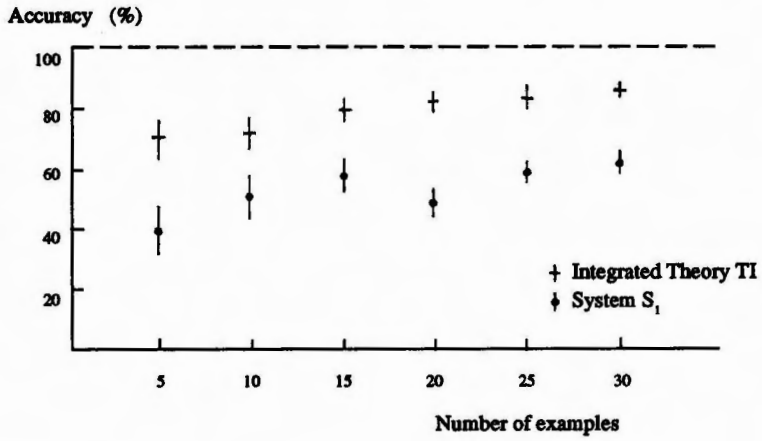



Fig 4. Performance of the Integrated Theory TI and System $S_1$. In this figure the performance of the integrated theory (TI) is contrasted with the performance of systems $S_1$. Standard deviations are indicated by vertical bars. The graph is based on the data shown in Table 1.

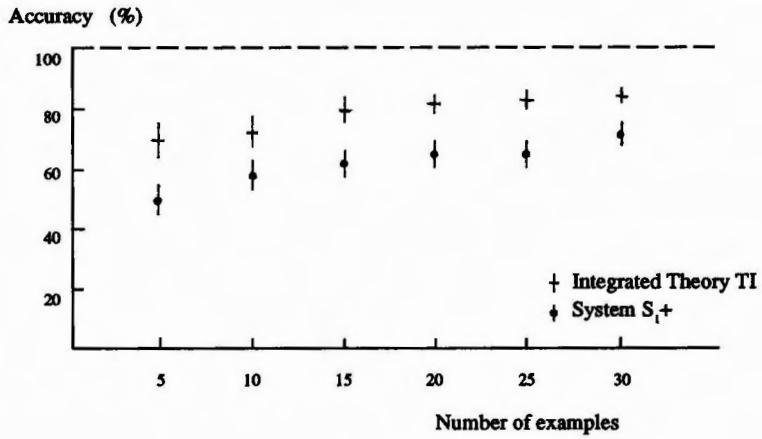



Fig 5. Performance of the Integrated Theory and System $S_1$+.

The results of these experiments have exceeded our expectations. We have expected the integrated theory to perform well, but the performance gains were really higher than we have expected. Had there been anything wrong with our experiments?

It is conceivable (although unlikely) that the present versions of ITL1 and IRL1 have a suboptimal performance. But this would not be an argument against integration. System INTEG.3 tries to make the best out of the theories it gets, irrespective of whether they are optimal or not. INTEG.3 checks them out and produces an integrated theory that, as we have seen, performs generally well.

The experiments confirmed our belief that it is worth pursuing this issue further. In the next section we discuss the relationship of this method to other work. We also describe some of the shortcomings of our approach and possible extensions that could be made in future.

# 3. Discussion

### Integration is a Form of Learning

The process of constructing of a theory from several individual theories can be regarded as a *form of learning*. The process can be regarded as *learning*, because the system's internal representation gets reformulated and this leads to an improvement of performance. We notice that the reformulations are rather simple. They involve inclusion (or non inclusion) of rules in the integrated theory and modification of coefficients affecting their selection. Consequently, we use the term *weak* form of learning to distinguish it from other more complex forms of theory reformulation which could also involve modifications of individual rules. Despite its simplicity, however, this method leads to noticeable improvements in performance. More work can, of course, be done in this area. In the following sections we will discuss some possible extensions of this work.

### Knowledge Integration and Redundant Knowledge

Gams [1989] has argued that redundant knowledge is essential for superior behaviour of learning systems and his measurements support this claim. Gams further justifies this approach as follows: *"It is commonly accepted that cross checking of several knowledge sources is generally better than using one source of knowledge alone."*

It is conceivable that INTEG.3 achieves higher performance than expected thanks to some kind of in-built redundancy mechanism. It is worth pointing out, however, that Gam's approach is different from ours. Gams attempts to represent the redundant knowledge in an *explicit* manner. Although INTEG.3 also exploits several theories in the process of constructing an integrated theory, there is no attempt to represent redundant knowledge within the system. The integrated theory has a similar structure as the original theories. Also, it is not much bigger in size than any of the original theories. Thanks to the fact that the integrated theory has a relatively simple form, it can be used as input in further learning.

### Analytic Versus Empirical Approaches

Our system could be extended to perform an analysis of the rules that have been obtained. The system could try to *merge* the existing rules and and *reformulate* them so as to avoid possible conflicts and overlaps, as was suggested, for example, by Steels and van de Velde [1989]. Our system constructs the integrated theory solely on the basis of empirical evidence. However, empirical approaches are quite powerful.

Consider, for example, the problem of determining whether one rule *subsumes* another. This is a difficult task in general. The failure to conclude the proof within a given time does not mean that proof will not possibly succeed later. If on the other hand one relies on an empirical approach, the task is somewhat easier. INTEG.3, for example, will analyse the given cases to get a rule that covers some of them. Then it will consider the cases that have not been covered yet, and so on. The chances that INTEG.3 would generate rules that subsume others seem to be small. However, if the domain were very noisy, it could perhaps happen that inconsistent rules would appear in the integrated theory. Of course, preference would always be given to the rule that is qualitatively better, so the problem would not really affect the system's function much. But one could argue that whenever the data is noisy, it is wrong to try to generate perfect rules that 'hide the problem under the carpet'.

### Knowledge Integration and Theory Revision

The BLIP system [Emde, 89] is capable of representing several *competing theories*. Emde points out that the decision whether some fact is consistent with the theory depends much on the viewpoint. A particular fact can contradict one theory, and at the same time, it can be consistent with another theory. Contradictions can thus act as a driving force for changing the individual theories. Each competing theory will then follow a certain evolution path. This process continues until, presumably, some theory is obtained that is found to be satisfactory. In BLIP no attention has, however, been paid to the issue of how different parts of theories can be combined. In our view this is important, as it can speed up the process of constructing the required theory.

## 3.1 Future Directions

### Retaining Justifications in the Integrated Theory

We notice that many recent ET tools (such as Aquinas) do not try to represent knowledge in *one* homogeneous knowledge base, but rather provide capabilities for representing *different views*. This argument seem to suggests that we *should not* really try to construct an integrated knowledge base. One could argue that integration might not really be desirable.

In our view this argument is false. Knowledge integration *does not exclude* the possibility of *keeping* the different views within the system. Keeping this information has the advantage that the system can *explain* its decisions. The system can tell us, for example, where the rule came from (i.e. from which system), and why it was included in the integrated theory. It is also possible to let the system revise its theories in the light of new experience or as a result of communication with other systems.

### Using Integrated Theory in Further Learning

One of our future aims is to extend the system so as to be able to improve the integrated theory after new information has become available. The system would then function as a 'closed loop system': it would be capable of using the integrated theory as input for further learning. But which part of the theory should we attempt to modify?

One rather obvious strategy tries to focus the attention on 'weak parts' of the theory so as not waste too much effort with unnecessary modifications. Identifying weak parts of the theory is a bit like debugging, only it is more complex. In debugging one tries to identify a faulty step, by comparing, for example, the user's answer with the system's answer. Here one needs to consider how a particular step contributes to overall system´s performance. If some particular step degrades the system's performance, it is a good candidate for further changes.

Murray and Porter [1989] have done some work in this area. They have developed a system called PROTOKI, which is a prototype of a larger system called KI. This system works as a knowledge integration tool. Its aim is to integrate a new piece of knowledge in the existing knowledge base. This process includes the following three main steps.

The first one is called *recognition*. This step is concerned with identification of the part of existing knowledge that could be affected by new information. The second step is referred to as *elaboration*. In this step the system tries to determine the consequences of new information and verify whether some anomalies could be detected. An anomaly will arise if conflicting conclusions could be reached using different chains of reasoning, or if the system's answer conflicts the answer supplied by the user. The third step is called *adaptation*. This step is concerned with the problem of resolving anomalies. The

prototype system inspects the explanation that led to the observed anomaly. Heuristics are used to determine what could be considered the "weakest premise" of the chain of reasoning, and this premise is then modified.

The system described uses a great deal of domain dependent knowledge in the process. There is no harm in using such knowledge - as long as the general principles are clearly spelled out. These have not been described in great depth. So far PROTOKI has been oriented towards one example. Also, it is not clear what should happen if the user did not quite agree with the changes made by the system.

### Overcoming Language Differences between Agents

Another line that could be investigated is how to overcome the problem of language differences between agents. This problem typically arises when knowledge is elicited from several (human) experts. Experts can disagree on the vocabulary and the meaning. As Shaw and Gaines [1989] have pointed out the *same* term can have *different* meanings for different systems. They call this situation a *conflict*. *Different terms* may, however, have *identical* meanings. Shaw and Gaines call this situation *correspondence*.

It is easy to see that INTEG.3 deals with the problem of *conflict*, but not of *correspondence*. That is, the system uses empirical evidence to decide whose definition of the given concept is 'best'. There is no contention as to the concept names. These are assumed to behave like *rigid designators* [Genesereth & Nilsson, 1987 (p.234)]. Work is under way to construct a system capable of overcoming the problems of correspondence. The prototype system that is described in [Brazdil et al., 1990] can learn to define the concept of one agent using other agent's terminology.

Some of these issues may be rather difficult to resolve. Different agents may not only use different predicate vocabulary in their conceptualization, but also rather *different conceptualizations* [Morik, 1989]. The AI community has been concerned with these problems, but no simple solution is in view. Obviously, we cannot provide a simple solution here, although we intend to work on some of these issues in future.

### Conclusion

In this paper we have discussed the method of knowledge integration and described the implemented system INTEG.3. Our experimental results have exceeded our expectations. The integrated theory had a significantly better performance than the original theories. This work suggests that the method could be incorporated into some of the existing knowledge acquisition tools. This would provide a way for integrating the knowledge obtained from experts with the knowledge obtained inductively on the basis of data.

## Acknowledgments

The authors wish to thank Katarina Morik for some useful comments during a seminar on this topic. These comments made us spell out the differences between knowledge integration and incremental learning with greater clarity. We would also like to thank various anonymous referees for their comments and suggestions. Finally thanks also to Jeff Schlimmer, Ming Tan, Tom Mitchell and others at Carnegie Mellon who have discussed this work with me during a recent visit.

This work was partly supported by *Project Ecoles* (Esprit 2 Project 3059). The authors wish to express their gratitude to the Commision of European Communities for this support.

## References


Bond A.H. and Les Gasser (1988): *Distributed Artificial Intelligence*, Morgan Kaufmann.

Cestnik, B., Kononenko, I. and Bratko, I., (1987): "ASSISTANT 86 : A Knowledge-Elicitation Tool for Sophisticated Users", in *Progress in Machine Learning*, I.Bratko and N.Lavrac (eds), Sigma Press, Wilmslow, England.

Boose John, Bradshaw J., Kitto C. and Shema D. (1989): From ETS to Acquinas : Six Years of Knowledge Acquisition Tool Development, in *Proc of Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, J.Boose, B.Gaines and J.G.Ganascia (eds), Paris, July 1989.

Brazdil, P. and Clark, P. (1990): "Learning from Imperfect Data", in *Machine Learning, Meta-Reasoning and Logics*, P. Brazdil and K.Konolige (eds.), Kluwer Academic Publishers.

Brazdil, P., Muggleton S. and Hoppe T. (1990): Learning to Relate Terms in Multi-agent Environment, Working Paper (in preparation), LIACC, University of Porto.

Clark, P. and Niblett, T. (1987): "Induction in Noisy Domains", in *Progress in Machine Learning*, I.Bratko and N.Lavrac (eds), Sigma Press, Wilmslow, England.

Clark, P. and Niblett, T. (1989): "The CN2 Induction Algorithm", *Machine Learning 3*, Kluwer Academic Publishers.

Emde W. (1987): "Non-cumulative Learning in METAXA-3", in *Proc. of IJCAI-87*, Milan, Italy, pp. 208-212.

Emde W. (1989): "An Inference Engine for Representing Multiple Theories", in *Knowledge Representation and Organization in Machine Learning*, K. Morik (ed.), pp. 148-176, Springer Verlag.

Gaines B. (1989): Knowledge Acquisition: the Continuum Linking Machine Learning and Expertise Transfer, in *Proc of Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, J.Boose, B.Gaines and J.G.Ganascia (eds), Paris, July 1989.

Gams M. (1989): "New Measurements Highlight the Importance of Redundant Knowledge", in *Proceedings of 4th European Working Session on Learning*, K. Morik (ed.), pp. 71-80, Pitman - Morgan Kaufmann.

Genesereth, M. and Nilsson, N. (1987): *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann Publishers Inc.

Janikow C.Z. (1989): "The AQ16 Inductive Learning Program: Some Experimental Results with AQ16 and Other Symbolic and Nonsymbolic Programs", Rep. of AI Center, George Mason University.

Morik K. (1989): "Sloppy Modeling", in *Knowledge Representation and Organization in Machine Learning*, K. Morik (ed.), pp. 107-134, Springer Verlag.

Murray K.S. and Porter B.W. (1989): "Controlling Search for the Consequences of New Information During Knowledge Integration", in *Proc. of 6th International Workshop on Machine Learning*, A.M. Segre (ed.), Ithaca, New York, Morgan Kaufmann Inc.

Quinlan J.R. (1986): "The Effect of Noise in Concept Learning", *Machine Learning (vol. II)*, R.Michalski, J.Carbonell and T. Mitchell (eds), Morgan Kaufmann Publishers, Inc.

Rada R. and Mili H. (1989): "A Knowledge Intensive Learning System for Document Retrieval", in *Knowledge Representation and Organization in Machine Learning*, K. Morik (ed.), pp. 65-88, Springer Verlag.

Shaw M. & Gaines B. (1989): Knowledge Acquisition: Some Foundations, Manual Methods and Future Trends, in *Proc. of Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, J. Boose, B. Gaines and J.G. Ganascia (eds), Paris, July 1989.

Steels L. and van de Velde W. (1989): "Learning in Second Generation Expert Systems", in *Machine and Human Learning*, Y. Kodratoff and A. Hutchinson (eds.), Michael Horwood Ltd.

Torgo L. (1990): "Incremental Rule Learning Using Entropy Based Search" (in preparation).

Utgoff P.E. (1988): "ID5: An Incremental ID3", in *Proc. of 5th International Workshop on Machine Learning*, J. Laird (ed.), Ann Harbour, Morgan Kaufmann Inc.

Van de Welde W. (1989): "Re(presentation) Issues in Second Generation Expert Systems", in *Knowledge Representation and Organization in Machine Learning*, K. Morik (ed.), pp. 107-134, Springer Verlag.

Wielinga B.J. & Breuker J.A. (1986): Models of Expertise, in *Proc. of ECAI-86*, L.Steels (ed.), Brighton.

Zhang J. and Michalski R. (1989): "Rule Optimization via SG-TRUNC Method", in *Proceedings of 4th European Working Session on Learning*, K. Morik (ed.), pp. 251-262, Pitman - Morgan Kaufmann.

# Producing Visually-based Knowledge Specifications for Acquiring Organizational Knowledge

James P. Davis
Horizon Research, Inc.
1432 Main Street
Waltham, MA 02154
jdavis@hri.com

Ronald D. Bonnell
Center for Machine Intelligence
University of South Carolina
Columbia, SC 29208
bonnell@ece.scarolina.edu

**Abstract.** Most work in knowledge acquisition assumes that there is a single problem-solving task for which the knowledge base being constructed through acquisition will be applied. However, there is growing interest in a class of knowledge-based systems, referred to as *organizational knowledge systems*, where it cannot be assumed that there is a single task model or problem-solving method for which the knowledge is to be acquired. This paper discusses this type of knowledge-based system, the factors in organizational problem-solving which mitigate its existence, and presents an approach to knowledge acquisition in this environment. The acquisition technique, known as *Conceptual Abstraction*, has its basis in both semantic data modeling and model-based reasoning. It has been effectively applied in the specification of *knowledge schemata* for organizational knowledge bases. The paper discusses the underlying model for abstraction and the associated abstraction mechanisms of the Conceptual Abstraction approach, and illustrates their use through example.

## 1. Introduction

There has been considerable success in recent years in developing tools and methodologies to alleviate the knowledge acquisition "bottleneck" associated with the construction of operational knowledge-based systems. A foundation for much of this progress has been realizing the importance of "knowledge level" consideration of the problem-solving in some domain at an abstract level, rather than at an operational level. This view of knowledge acquisition emphasizes constructing abstract descriptions of the problem-solving *tasks* themselves, and defining the appropriate *methods* for automating the problem-solving for these tasks [12] [40] [41] [44].

This approach to knowledge acquisition for knowledge-based systems assumes that the acquisition activity is carried out relative to problem-solving method(s) that are defined "a priori" for the domain. The abstract description of the problem-solving process(es) is generally encoded into the target "task-performance" environment (such as a task-specific knowledge base shell). Furthermore, this representation of the problem-solving process is used to constrain the construction of task or domain models in order to operationalize the problem-solver for the problem at hand.

However, there is a recent trend in knowledge-based systems (due in part to the confluence of ideas between the AI and database communities) towards *organizational knowledge* (or, knowledge-based) *systems*. Organizational knowledge systems are based on the premise that there exists a single knowledge store that may be operated on simultaneously by many different and diverse problem-solvers and reasoning engines. Because there is a decoupling of organizational knowledge from knowledge of the problem-solving tasks which utilize it, the knowledge acquisition problem must be examined within a framework that assumes independence (and some

degree of ignorance) of the abstract (and operational) problem-solving methods and tasks to which the organizational knowledge will be directed[1].

This paper presents such a knowledge acquisition framework known as *Conceptual Abstraction,* in terms of its underlying model and abstraction mechanisms, and demonstrates the use of this approach in the construction of knowledge schemata as an intermediate knowledge-level specification of the acquired knowledge. This specification, represented (either "on paper" or "on screen") through use of a graphical (or "visual") representation technique, can be subsequently mapped into appropriate symbol-level structures (in some knowledge representation language) for use by problem-solving engines defined external to the organizational knowledge management environment.

The process of constructing *schemata* specifications as a means of representing domain knowledge for some enterprise during its acquisition for organizational knowledge systems is discussed in this paper. The approach presented is based on the authors' work in creating an environment and mechanisms to support a Knowledge Base Management Systems (KBMS) architecture consisting of a shared knowledge repository used by multiple rule-based expert systems[2] [21] [22].

The paper puts forth the following line of reasoning. First, the problem of controlling the acquisition of *conceptual* knowledge (concept entities, relations, properties, and constraints) for constructing organizational knowledge bases is similar to that associated with constructing semantic models for databases. A semantic data model (which has been sufficiently extended with additional representational primitives) can be used for creating intermediate specifications, known as *knowledge schemata*, that can be mapped to either database or knowledge-base environments [19] [21] [56] [24]. Second, the "behavior" of the modeling framework itself (in terms of its semantics) can be used to control many aspects of the acquisition process itself (referred to as *model-based knowledge acquisition* [44] [28]). Third, consideration of concept acquisition as a first step in the construction of knowledge-based systems has proven to be an effective means for structuring resultant organizational knowledge bases that are "operationalized", once the initial acquisition process is complete.

The paper is organized as follows. Section Two discusses the nature of organizational knowledge and the characteristics of organizational knowledge systems. Section Three discusses the aspects of domain modeling and abstraction, and the relationship to knowledge acquisition. Section Four presents the Conceptual Abstraction framework, including the semantic model and the abstraction mechanisms supported in the model to construct knowledge schemata. Section Five presents this approach to knowledge acquisition in practice through discussion of a real-world example. Section Six summarizes the results presented in the paper and discusses current directions of the research.

---

[1] The precedence for this view of independence is primarily drawn from the Database field, where the notion of *logical data independence* has been understood for some time. This notion has been extrapolated into the realm of AI-Database systems as *knowledge representation independence* [62]. A similar idea has been expressed by Anderson in dealing with the characteristics of declarative knowledge [4]. However, as in modeling for database design, this doesn't preclude considering the effects of data organization on the ability to perform queries. Thus, decoupling the main knowledge acquisition process from the underlying problem-solving activities doesn't preclude considering the effects of a selected knowledge organization on the ability to support acceptable reasoning behaviors for the problem-solving tasks at hand. However, the implications of this statement are not addressed further in this paper.

[2] This paper doesn't attempt to justify the rationale for adopting a KBMS strategy for organizational knowledge systems, as this is presented elsewhere [21]; however, the paper does discuss the issues of acquisition in this environment. In addition, the paper does not attempt to define the architecture of a knowledge acquisition tool for this environment, as this is also discussed elsewhere [24]. Regarding the abstraction mechanisms to be discussed in this paper, there is no discussion of either the formal definition of the semantics of the mechanisms or of the computational framework in which these mechanisms are automated, as they are also being presented elsewhere [19] [20]. Finally, though this paper mentions that the schemata produced using this approach can be mapped to appropriate "symbol-level" structures, a discussion of the mapping process itself is outside the scope of this paper.

## 2. Organizational Knowledge Systems

### *2.1. The nature of organizational knowledge*

The increasing pace for infusing knowledge-based systems technology into the information management environments of corporate entities has been viewed as essential to a corporation's competitiveness [27]. Companies are posturing themselves to compete as "knowledge-based businesses", and are beginning to perceive their information, in-house expertise, and "corporate memory" as *organizational knowledge*, an important asset that must be structured, managed, maintained, and protected.

The nature of knowledge in a corporate entity (such as a computer manufacturing company) is usually such that it is created in one organization and is moved across organizational boundaries to be used elsewhere. This is illustrated logically in Figure 1, which depicts knowledge being created within Engineering and Manufacturing organizations for use in the manufacturing and the subsequent field support of computer system products.

Relative to the use of knowledge to leverage and amplify the performance of specific tasks, companies are employing knowledge-based systems at varying points in their overall organizational processes. The problem-solving activities at these different points require problem domain-specific and task-specific KBS solutions. The current philosophy of developing knowledge-based systems tends to consider each of these problem-solving tasks independently of one another[3].

In organizations which employ KBS solutions at various points in their processes, considerable overlap in the knowledge contained in the KBS may exist. Companies that have deployed many of these KBS solutions are beginning to grapple with the enormous problem of maintaining these systems individually as they evolve over time [52]. Some companies are attempting to manage a different problem of maintaining several KBS's simultaneously--where these systems have been shown to contain overlapping domains of knowledge that are used in different problem-solving tasks in a larger process [21].

### *2.2 The characteristics of organizational knowledge systems*

There is a recent trend in knowledge-based systems (due in part to the confluence of ideas between the AI and database communities) towards *organizational knowledge* (or, knowledge-based) *systems*. This type of system may also be referred to as a type of *knowledge base management system* [11] [21] or *expert database system* [35]. Put simply, an *organizational knowledge system* is a system that provides management of a large shared knowledge base for (potentially) many knowledge-based applications.

Due to the nature of organizational processes, and the desire to exploit knowledge across organizational boundaries, organizational knowledge systems must have the following characteristics: *large-scale* (assuming characteristics of both *knowledge*--with a large number of structured classes, concepts, relations, and constructs denoting "behavior", and *data*--with a large number of regularly-structured records); *shared knowledge* (supporting the maintainability of corporate knowledge required in a large organization); *multifunctionality* (supporting knowledge being used for problem-solving in multiple problem domains employing different reasoning methods); *persistence* (being able to retain what has been concluded by a KBS in problem-solving); and *direct access to live data* (most industrial-strength KBS applications are reasoning about problem domains where the "facts" are based on the contents of existing on-line information systems).

Many of these limitations, as they exist in current KBS technology, become the basis for adopting a KBMS architecture. The nature of the knowledge acquisition problem for creating organizational knowledge bases for this type of knowledge-based systems environment must take these characteristics into consideration.

---

[3] This does not account for current research into distributed AI (DAI) systems [32].

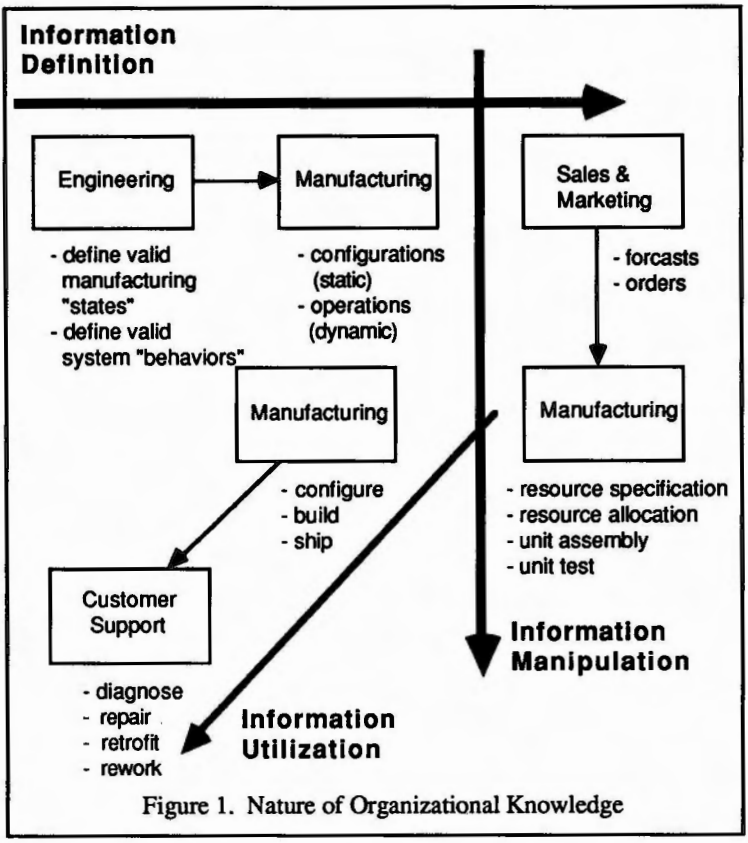



Figure 1. Nature of Organizational Knowledge

## 3. Domain Modeling, Abstraction, and Knowledge Acquisition

### *3.1. Confluence of Research Domains*

There has been a recent resurgence of interest in domain modeling as a basis for formulating specifications of systems, whether the systems are conventional software-based [59], database [45] [10], or knowledge-based [42] [44] in nature. In this view, creating abstract specifications for such systems not only provides a useful intermediate form in which to represent knowledge about a domain of discourse associated with a system, but also provides an effective means for better interaction between the "informant" (whether system user or domain expert) and the "engineer" (whether of the software, database, or knowledge variety).

Furthermore, there has been much effort in the AI, Database, and Cognitive Science communities directed towards defining mechanisms for specifying the structure and behavior of a domain in order to construct models of a domain at some acceptable level of detail [6] [33] [53] [38] [60] [19]. Much of the work of interest has been in defining appropriate abstraction mechanisms which can be used to formulate domain "theories". This is motivated by the notion that much of what is perceived in the world is generally well-structured information (data, knowledge), and that a large problem in constructing complex systems capable of intelligent behavior is in elucidating these conceptual structures through appropriate abstraction mechanisms.

Representing these evolving specifications in a form that is both computationally tractable and cognitively intuitive has also been a focal point of much research. Much of the work in the

Database community in *semantic data modeling* [30] [55] has focused on creating graphical representations of semantic models (known as *schemata*[4]) for arbitrary enterprise domains. Modeling techniques such as the Entity-Relationship approach [14] and its variants have proved useful in this capacity[5]. It is interesting to note that using semantic models for constructing schemata for knowledge-based systems has been applied in practice [26] [56] [24].

Finally, an area of research which is providing some interesting direction in creating intuitive metaphors for expressing domain information is work in the *visual languages* community [57]. In this research area, work is being done to create languages for expressing information at conceptual levels without resorting to text-based languages (whether they are "natural languages" such as English or German, or symbol-level "computer languages" such as Pascal, SQL, or KRL).

### *3.2. Domain modeling and knowledge acquisition*

Constructing models of a problem domain is recognized as an important component of the overall task of knowledge acquisition for expert systems [2] [3] [48] [42] [44] [47]. As discussed earlier, it is usually carried out relative to some specific problem-solving framework.

However, there are several approaches to domain modeling for purposes of knowledge acquisition that are carried out independent of some pre-defined problem-solving model. These generally involve creating taxonomic models, or *ontologies*, of artifacts existing in the domain of discourse [3] [48] [39]. These models are constructed either independent of, or as a precursor to, more extensive analysis of the problem task itself.

This process of constructing ontological models of a domain emphasizes eliciting terminology and concepts from an expert or "informant", and organizing them into conceptual structures which accurately represent the view of the domain as perceived by the informant. The underlying acquisition techniques focus on refining the constructed models as a basis for applying other processes to obtain complete domain descriptions--including ontologies, task descriptions, knowledge sources, and problem-solving strategies [8].

### *3.3. Abstraction and Domain Modeling*

The basic process whereby a "model" of the real-world is constructed is known as *abstraction*. There are many techniques and mechanisms allowing a model or representation of some view of the real-world to be constructed. The underlying principle of abstraction involves the *selective emphasis of detail*. Simply put, this is knowing "what" is important in the real-world that needs to be modeled at any given time, from any given point of view, as well as knowing "when" to refine portions of the model as it evolves.

The end result of applying abstraction techniques is that a model specification (or *schema*) exists which accurately "stands for", or represents in some meaningful way, those aspects of the real-world that are of interest to the human "observer". Generally, the model created may emphasize certain characteristics of the world as perceived by the observer, and may organize it in ways that may not be readily apparent in reality (at least to someone not intimately familiar with that "world"). Abstraction mechanisms provide the conceptual "tools" for realizing acceptable models of reality. These tools involve applying specific mechanisms that allow descriptions of the world to be constructed having a high level of structure and organization. Elucidating that structure and

---

[4] A *schema*, as used throughout this work, is a "specification", or "plan", for organizing information in terms of the classes (or entity sets), relationships, and constraints that specify the appropriate semantics to sufficiently constrain the set of valid states conforming to (or denoting) the specification.

[5] It should be noted that the Entity-Relationship model has been used extensively over the past fifteen years in logical database design (to the point that it is a standard component in most commercial CASE tools). Most applications of the E-R model have been as an intermediate step in the database design process, where E-R models of enterprise schemata are mapped into an underlying database model, such as the relational model. However, there is now a great deal of interest in using the E-R model as a knowledge representation formalism in its own right [50]. It has recently been used in such areas as image analysis [37], intelligent planning [31], intelligent tutoring [54], expert systems design [26] [36] [7], and knowledge acquisition [44] [56] [24].

organization of what is "known" into a precise model forms the bulk of what the abstraction process is all about.

The use of abstraction techniques to incrementally build a schema tends to reduce the complexity of large schemata, as well as eliminate some forms of ambiguity that would otherwise result from incomplete specification of domain semantics. In this sense there is less in the domain that is "not said"--where specific assertions which, due to their absence, can be ambiguously interpreted as either being true or false.

### *3.4. Abstraction and knowledge acquisition*

Creating abstract representations of a "reality" that is perceived by an informant is a basis for knowledge acquisition via model construction. Generally the medium for capturing the appropriate knowledge for constructing models is accomplished through text analysis to identify terminology and concepts [8] [48] [39]. There are other approaches which use text-based modeling frameworks to allow the expression of knowledge in those terms for constructing models [3].

However, from work in the visual languages field [17] [43] [57], it is possible to define visually-based modeling frameworks in which to express knowledge about the perceived domain of discourse. The fundamental assumption is that the manner in which concepts from the domain of discourse are expressed by the informant during acquisition will be directly shaped by this language [58]. Thus, rather than expressing what is "believed" about the world in terms of natural language constructs, this expression of belief is defined in terms of *categories* existing in the visual language, where these categories are abstractions of categories existing in natural language[6] used in perceiving phenomena in the real world.

### *3.5. The premise of Conceptual Abstraction*

The knowledge acquisition framework used in the creation of organizational knowledge bases is known as *Conceptual Abstraction*. It involves the iterative application of abstraction and refinement mechanisms to create a method and task-independent knowledge schema. This knowledge schema specifies the structure and semantic constraints associated with an organizational problem domain.

Conceptual Abstraction does not replace approaches for acquiring knowledge about problem-solving behavior for any given task that is being automated with components of an organizational knowledge system. It also does not replace techniques to aid in basic elicitation of characteristics of the domain (such as repertory grid, verbal protocol analysis, etc.).

The premise of Conceptual Abstraction is similar to that of Ontological Analysis [2] [3], Frame Classification in the KREME system [1], and Systemic Grammar Networks [34] as a set of tools and conceptual frameworks for guiding certain aspects of the elicitation process in knowledge acquisition. Conceptual Abstraction provides a modeling framework that is used as a means to structure the acquisition of domain models through the systematic application of abstraction and refinement processes. It also serves as a "mediating representation" [34] between collections of verbal data and symbol-level knowledge representations.

## 3. The Conceptual Abstraction Framework

The framework for Conceptual Abstraction consists of the following components: a set of modeling primitives (constituting a modeling language) and associated graphical constructs used for constructing knowledge schemata of some domain of discourse; specific abstraction

---

[6] The notion of categories from natural language implies expressing what is known in the world in terms of words classified into natural language categories (nouns, verbs, adjectives, etc.) that are combined into sentences according to a grammar indicating "well-formedness". For abstract modeling frameworks that are defined from a language perspective (such as visual languages), the categories defined in the language for classifying conceptual "words" are based on how the world is abstracted. In the approach defined in this paper, the categories correspond to a set-based abstractions including entities (corresponding to nouns), relationships (corresponding to verbs), and other constructs. The association between set-based constructs and English categories is established in [15].

mechanisms and conventions of use for constructing well-formed "theories" using the vocabulary of the model to express the terms and concepts of the domain; and, guidelines and a methodology for using the model and the abstraction mechanisms in the acquisition activity. Since the focus of this paper is on the use of the abstraction mechanisms themselves (which are the modeling primitives combined in ways to express the assertions that are part of the domain model), the methodological issues are not covered here, but are addressed in [24].

### *3.1. Model for Conceptual Abstraction*

The modeling framework used with Conceptual Abstraction is based on a semantic framework known as EARL[7] [19] [20] [21] [23]. This modeling framework was created to support capture and expression of complex semantics of database and knowledge-base schemata. It is based primarily on the Entity-Relationship model [14], but has been enhanced with mechanisms from more robust knowledge representation formalisms[8].

EARL adopts an abstract view of the "world" where artifacts perceived in the world are described in EARL according to the vocabulary shown in Figure 2 (depicted as an ontology). Everything that is perceived is expressed as being a either a *set* of things (containing unique and identifiable instances), *function* mappings between these sets (as between a domain and co-domain), or a *constraint* on assuming a certain "state of being" by artifacts existing in this set-based abstraction space. These specialized types of artifacts are further classified into sub-categories (which are not shown here, but described in [21]). The basic artifacts which serve as the vocabulary for expressing domain "theories" are the *entity set*, the *relationship set*, the *attribute*, and a taxonomic assortment of *semantic constraints*. Individual instances of perceived phenomena are categorized according to this vocabulary.

This process of abstracting the real world into some representation is based on the classification of domain artifacts into sets, where set members hold certain common properties. EARL exploits this view by defining everything in terms of such sets. The functions existing between sets are binary mappings that occur between them. The constraints existing are analogues to those that exist in the perceived real world, but are expressed in EARL as restrictions on the nature of the mappings between the various types of sets in the schema abstraction.

An additional aspect of EARL is the existence of a "meta-schema" that contains a meta-model of the vocabulary and grammar of EARL itself. Its characteristics are that it is declarative (allows EARL itself to be interpreted propositionally by the "reader"), explicit (allows the modeling framework itself to be inspectable, pointed out as a limitation in earlier work [3]), and self-describing (in that EARL structure and behavior is capable of being expressed as an EARL model of itself). The utility of this capability, as well as a complete definition and ontology of EARL is discussed elsewhere [21].

### *3.2. Abstraction mechanisms*

There are several different but related perspectives on concept abstraction, coming from semantic data modeling [30] [45] [51] [55], knowledge representation [1] [5] [6] [13] [29] [38] [46] [53] [61], and cognitive science [4] [33] [60] fields. Specifically, these notions deal with the types of abstrac-tion mechanisms available, and for what purpose they are used in representing the semantics of some domain of discourse.

The three major types of semantic abstraction mechanisms defined for use in knowledge acquisition are: *generalization/specialization* [51] [29] [55] [6] [53], *aggregation/decomposition* [51] [55] [30] , and *classification/denotation* [9] [46] [30]. Each form of abstraction is discussed in the following paragraphs.

---

[7] EARL stands for Entities with Abstraction and Relationships with Logic.

[8] The objective of the EARL framework is not to serve as an all-encompassing knowledge representation (as is the focus of most AI research into knowledge representation); it has, however, been devised to serve as a basis for creating abstractions of the real-world that are of some interest, and that are amenable to capture in terms of a set-based vocabulary. A set-based vocabulary is used as opposed to the role-filler vocabulary of representations such as KL-ONE [5]; a comparison of EARL and other representations is topically presented in [20] [21].

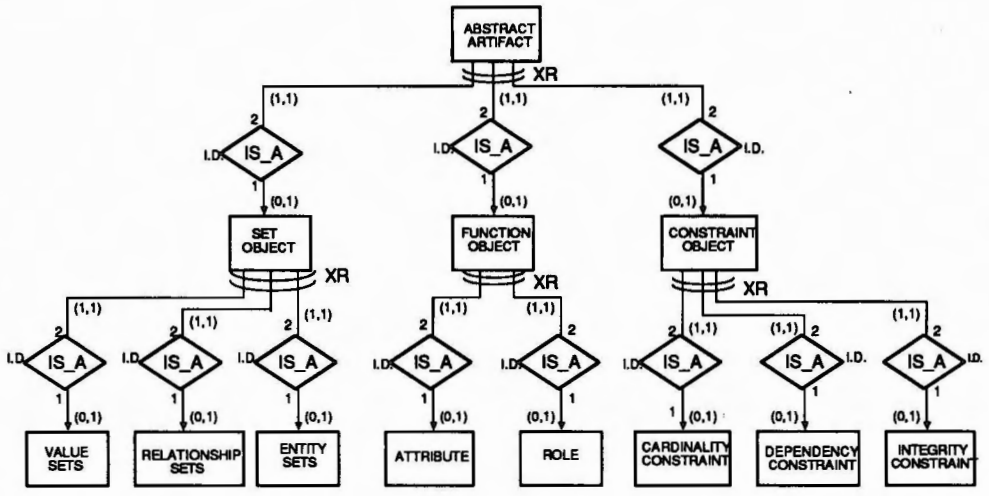



Figure 2. Meta-level Ontology of EARL

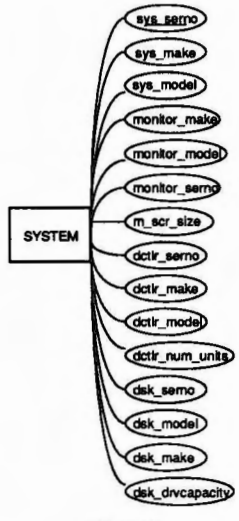



Figure 3a

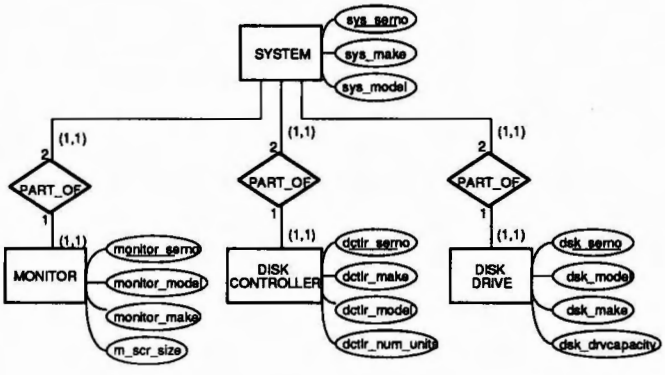



Figure 3b

Figure 3. Part-Whole Aggregation Abstraction

### *3.2.1. Generalization*

*Generalization* is the basic form of abstraction used in the placement of structure and organization onto a representation of some domain. This abstraction relation specifies a *type-type* relationship between an object and its constituent objects, placing emphasis on the similarities of objects while abstracting their differences down into the hierarchy.

The use of *roles* in the context of generalization type hierarchies has been documented in the literature [55] [53] [20]. This is based primarily on the conceptual distinction between "natural" entities (which stand for some real-world object or thing) and entities which are capable of having relationships with other entities in the domain based on the "role" that they assume in the context of the relationship.

EARL provides the IS_A generalization relationship to represent three different notions of the IS-A generalization/specialization relation (as defined in [6]): the *subset/superset* IS-A, the *generalization/specialization* IS-A, and the *A-KIND-OF* IS-A relations[9].

EARL employs the distinction between "natural" and "role" types [53] through supporting the ROLE_OF generalization relationship. This is a *type-type* form of generalization where instances in an entity set may specialize into another entity set on the basis of some "functional" role the instances may play in the context of some relationship with other entities in the domain. The ROLE_OF construct expresses the semantic relation of "non-kind" status, in that an entity may assume a "role" (i.e., some "artificial" or "situational" state of existence) according to relationships that it participates in with other entities in the domain of discourse.

### *3.2.2. Aggregation*

*Aggregation* implies that an object has components as part of its structure that are themselves objects of interest in the domain of discourse. Aggregation is useful as an abstraction mechanism in that it gradually makes visible the composition of an object, and depicts how the components which comprise the object relate to each other and the object as a whole.

EARL makes a distinction between four types of aggregation, combining notions of aggregation and PART-OF from knowledge representation and data modeling. The first deals with the definition of a concept in terms of its internal structure. A type description is formed from the aggregation of properties and parts into the whole [29] [5] [49] [60]. This is a relationship between a particular kind of whole ("integral object") which is divided into components, where the components have specific structural and functional relationships to one another and to the whole object. The PART_OF relationship in EARL expresses this *component-integral object* relation.

The second notion of aggregation facilitates concept definition in terms of constituent subsets combined to form the description of the whole set [13] [60]. This type of aggregation usually forms a partition over the constituent subsets. The SUBS_OF relationship in EARL expresses this *member-collection* relation[10].

---

[9] This apparent "semantic overloading" of the IS_A relationship in EARL is done for two reasons. First, there is a "duality" of *set* and *type* behavior of entity sets in EARL. Entity sets are defined over some domain of discourse; however, the definition of these sets, in terms of their denotation, is based on predicates which define a type. The predicates might be defined over attributes of the entity set (specifically the non-null and "identifier" attributes, which ensure uniqueness and identity for the member/instances of the set/types). Second, there is no special distinction for "kind" status (as defined in [6]) in IS_A taxonomies.

[10] It should be noted that there is a subtle distinction between the semantics of the IS_A and SUBS_OF abstraction mechanisms. The IS_A relationship captures the *type-subtype* generalization semantics [6] (which implies a downward inheritance in the hierarchy) whereas the SUBS_OF relation-ship captures the *set-subset* aggregation semantics [13] (which implies upward inheritance of properties in the hierarchy). The closeness of the concepts of *type-subtype* and *set-subset* is such that they are sometimes difficult to distinguish [6] [49]. EARL provides abstraction constructs for both semantic situations because of the inherent set/type duality that is observed. Having specific abstraction constructs provides for some delineation and distinction between these two semantic notions, and how they are used in modeling the domain of discourse. This is discussed further in [21].

The third aggregation notion involves specifying concept descriptions based on the collection of properties defining the type, differentiating instances of the type from instances of other types by the property definition. The PROP_OF aggregation construct allows the properties of an entity set/type to be "projected" out of the type description. This expresses the *attribution* relation [60].

The fourth notion of aggregation deals with the formation of a higher level object based on the structure and relationships among constituent objects [51] [30]. The aggregate is defined not for the specific purpose of concept/class definition (in terms of internal structure, constituent sets, or property description), but for the purpose of abstracting away details of object behavior which are not relevant to a certain "view" of interest in the domain. This is an aggregation of entity sets/types and the relationships between them into higher level aggregate entity sets. The AGGR_OF relationship in EARL expresses the *relationship-abstraction* aggregation relation.

In other models and representations, there is generally no distinction made between the different semantic situations for which aggregation abstractions can be used. The four aggregation relationships are defined in EARL in order to provide a finer granularity for capturing subtle semantics in concept hierarchies.

### *3.2.3. Classification*

Classification is a form of abstraction in which a collection of objects is considered as a more abstract class to which the objects belong. The objects being classified into the higher level class may themselves be *instances* or *classes*.

The most prevalent form of classification is the INSTANCE-OF relation, whereby instances of objects are related to their class definitions (the *token-type* relation) [10] [46].

A second notion of classification relates types or classes to higher level types or classes to which they belong (the *type-metatype* relation) [55] [46]. These higher level classes/types are referred to as *meta-classes/meta-types*. This form of INSTANCE-OF classification views the lower level class(es) as instance(s) of the higher level class. From the perspective of the higher level class, all of the lower level classes form the *denotation* of the set of instances which belong to it [53].

A third notion of classification is represented by the MEMBER-OF relation. This form of classification relates the instances of a lower level type to the instances of the higher level type containing the classes to which the instances of the lower level type belong (a *type-type* relation).

EARL defines two classification abstraction relationships: the MEMBER_OF for expressing relationships between sets of instances and sets of classes containing them; and, the INST_OF for expressing relationships between individual domain instances that are of interest and the entity sets/classes to which they belong, and for the relationships between entity sets/classes and the meta-classes (defined in the EARL meta-schema) to which they belong.

## 4. Conceptual Abstraction in Practice: an Example

The use of the EARL abstraction mechanisms in the schema design process provides a set of principles for the iterative evolution of knowledge schemata for some domain of discourse that can be effectively mapped to "symbol-level" data and knowledge representation systems. This section presents an example of how the EARL abstraction mechanisms, coupled with mechanisms for specifying semantic constraints in EARL, can be effectively applied to guide the construction of a knowledge schema for a portion of the organizational knowledge system application in the domain discussed below. The example includes how one of each of the abstraction types is used in constructing domain models. Only the use of the abstraction constructs in creating domain models is discussed in this paper; the underlying model-based reasoning mechanisms required for automating the model construction process are discussed elsewhere [25].

### *4.1. The Enterprise Application*

The organizational knowledge system to be discussed exists in a computer manufacturing enterprise. As shown in Figure 1, there are three main organizations of interest: engineering,

manufacturing, and customer support. The application environment for this system includes several expert systems (built in commercially available expert system "shell" tools) that are tightly coupled to several database systems. Information about the following domains of interest: *computer order validation* (knowledge about the valid configurations which are supported for the population of hardware systems which this computer manufacturer sells and supports), *computer hardware problem diagnosis* (knowledge about specific behaviors associated with device and component failures, relating observable problem symptoms with problems and problem classes defined a priori[11]), and *computer population analysis* (knowledge about specific system configurations that exist at each customer site, including information about hardware and software revision levels, types of peripherals connected, and any revisions or updates to a system due to retrofit or repair).

This information must be defined at the engineering facility and provided from the company's manufacturing facilities, yet must also be maintained based on the service policies of this company on its customer's systems. The expert systems contain the heuristics of looking at patterns of behavior contained in the databases to infer trends in system operation (to apply towards preventive maintenance), and to determine cause of failure (to apply towards remedial maintenance).

### *4.2. Abstraction using Aggregation of Parts*

The process begins with some statement of the set of "relevant" entities, properties, and relationships which exist in the domain of discourse. This includes information about the structure and organization of the various components of computer systems. Figure 3a depicts an initial view of the information which might initially be deemed as relevant.

In this figure, there is an entity set SYSTEM which has a collection of properties associated with it. One of these properties, **sys_serno** is designated as the "prime attribute" (that attribute which acts as the identifier for instances in the set of systems in the domain of discourse), providing unique identification for instances of computer systems in the field. This entity set acts as an aggregation or collection of properties into a set description using the properties as "predicates" for membership in the entity set by instances in the domain of discourse on which the properties hold.

However, based on the semantics of the properties in their relationship to the entity set/class that they define[12], and based on heuristics of *functional dependency* and *normalization*[13] from the relational model [18], it is determined that it would be better to decompose the SYSTEM entity set into an aggregate collection of entity sets (MONITOR, DISK_CONTROLLER, and DISK_DRIVE). Intuitively, all of these properties, though associated with the aggregate description of what is a "system", have a better interpretation being associated with their own type/sets at a lower level of abstraction. This abstraction does several things:

---

[11] Though the organizational knowledge system construction process itself is carried out independent of the problem-solving tasks (which will use the knowledge) and the reasoning methods that are used in problem-solving, the framework for gathering and structuring knowledge can certainly be implicitly directed by some understanding of how the knowledge will be used, where that understanding is present. For example, the Incidents database [21] for this application has been defined based on the problem-solving model of differential diagnosis, the basis of heuristic classification [16]. The guidance for constructing the knowledge schema for this portion of the organizational knowledge base is not explicitly directed by the heuristic classification problem-solving model. However, since the schema was being developed for the purpose of declaratively specifying the "device/component" behaviors, this problem-solving model implicitly affected what kind of schema structure was appropriate.

[12] The semantics of how properties associated with a class description define that class are "intuitive", and are defined by the "observer" who perceives the view of reality which is being modeled.

[13] The notions of functional dependency, normalization, and multivalued dependency--concepts which are well-defined in the Relational Model--have been shown in EARL to be useful heuristics for controlling complexity and ambiguity in the abstraction process. The use of these heuristics is presented in [21], and explored further in [25].

1. It reduces the complexity of the concept description for a "system" by emphasizing the structured aggregate nature of what it "means" to be a system (as being the aggregate of its system components, such as disks, etc.); and,

2. It alleviates the problems with schema integrity which result from all of the properties not being dependent on the "identifier" for instances in the set which conform to the type description for a SYSTEM (groups of the properties are dependent on their own identifiers, such as **dsk_serno** and **dctlr_serno**).

These decomposed entity sets are related to the SYSTEM entity set through specific PART_OF relationships. Each of these entity sets has its own identifier, based on the component nature of what has been abstracted into the part-of sets. The semantic assertions for this schema portion are stated below, as depicted in Figure 3b.

each MONITOR is PART_OF exactly_one SYSTEM (1,1)
each SYSTEM has_as PART_OF exactly_one MONITOR (1,1)
each DISK_CONTROLLER is PART_OF exactly_one SYSTEM (1,1)
each SYSTEM has_as PART_OF exactly_one DISK_CONTROLLER (1,1)
each DISK_DRIVE is PART_OF exactly_one SYSTEM (1,1)
each SYSTEM has_as PART_OF exactly_one DISK_DRIVE (1,1)

The interpretation of this schema is based on the understanding that the sets represent actual components of real systems in the field. Each system has a terminal "monitor" (a system console, which serves as the primary terminal for software installation, etc.) a "disk controller" (to which disks are attached), and exactly one "disk drive". Each instance of these are identified with a serial number that is assigned by the component manufacturer. The attributes associated with each PART-OF set are functionally dependent on the value of the serial number acting as each set's identifier. The PART_OF aggregate relationship is selected to point out the *component-integral whole* relation that a "system" has with its component subsystems.

*4.3. Abstraction using Generalization/Specialization of Subtypes*

A generalization hierarchy is shown in Figure 4, where the CONTROLLER, SYSTEM, CABLE, and PERIPHERAL entity sets (some of which were introduced in Figure 3) are generalized into the DEVICE entity set. The PERIPHERAL subtype is further specialized into DISK, PRINTER, and TERMINAL subtypes. The set/types are specialized through the IS_A relationships. The semantic assertions of these IS-A specializations are defined as follows.

each DISK IS_A exactly_one PERIPHERAL (1,1)
each PERIPHERAL IS_A (may_be) at_least_zero and at_most_one DISK (0,1)
each PRINTER IS_A exactly_one PERIPHERAL (1,1)
each PERIPHERAL IS_A (may_be) at_least_zero and at_most_one PRINTER (0,1)
each TERMINAL IS_A exactly_one PERIPHERAL (1,1)
each PERIPHERAL IS_A (may_be) at_least_zero and at_most_one TERMINAL (0,1)
a PERIPHERAL may_be a DISK, or a PRINTER, or a TERMINAL, but must_be
one_and_only_one of these

The purpose of applying this abstraction mechanism is to organize the schema such that the properties that are shared by all the entity sets are abstracted to the minimally general level in the taxonomy. Thus, the properties associated with all the sets (the "serial numbers", "make", and "model numbers"--which are characteristic of all computer system components in the domain of discourse) are indicated at the DEVICE entity set, and are inherited by all other entity sets in the generalization hierarchy. Those properties which are not common among entity sets are abstracted downward to the entity sets to which they are associated.

The additional constructs that exist across the relationship links on the DEVICE, CONTROLLER, and PERIPHERAL entity sets are defined as *interlink semantic integrity constraints* [21] [23]. The purpose of these constraints is to capture behavioral restrictions that exist between an entity set and the many relationships through which it is related to other entity sets. The *XR Integrity* constraint defines a partition on the constituent specializations (as exist for DEVICE and PERIPHERAL entity sets), such that participation of an instance in the IS_A specialization

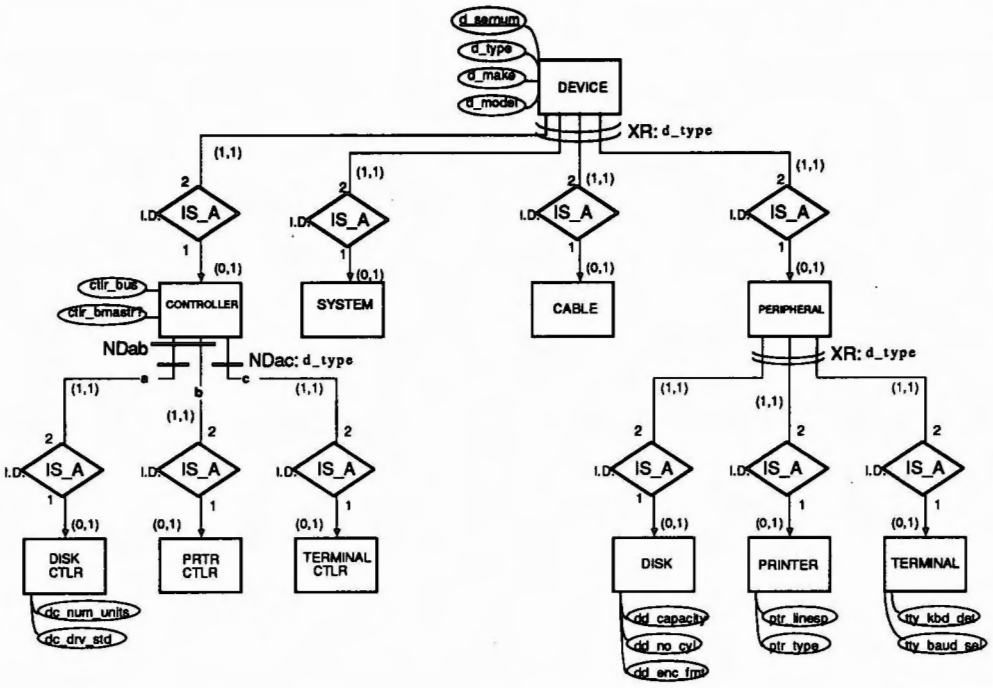



Figure 4. IS_A Generalization Abstraction

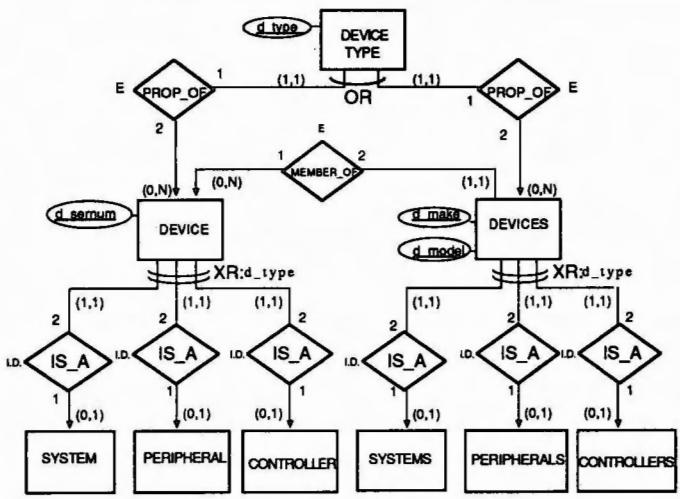



Figure 5. MEMBER_OF Classification Abstraction

relationship is *mutually exclusive* (a DEVICE may not exist as any more than one of the specializations) and *collectively exhaustive* (every instance of DEVICE in the domain must assume at least one of the specializations). The *ND Integrity* constraint (for example, the NDab constraint across the relationship links from the CONTROLLER entity set in Figure 4) asserts that "controller" instances may specialize into either "disk controllers" or "printer controllers", or neither (which might imply that the controller is a "terminal controller"), but may not specialize into both "disk" and "printer" controllers simultaneously (the semantics of this statement is that a particular controller board can't control these two types of devices, presumably because their hardware interfaces are incompatible). These constraints are based on propositional logic, and are further defined in [23].

### *4.4. Modeling Instances and Prototypes with Classification*

In this ongoing example, some views of the representation of the domain have been with respect to specific system/device/component "instances". Figures 3 and 4 dealt with this view of the domain of discourse. The instances are devices (such as disks, printers, and terminals) which are distinguishable from one another and identifiable from some property which guarantees uniqueness; this property is some "serial number" for a device, which is assigned and tracked during the manufacturing process for each and every component of a device or system of interest in the domain of discourse.

However, there is a different view of the enterprise which also must be modeled. This involves representing portions of the domain in terms of system/device/component "classes". These are the classes of devices (such as "disks" and "printers") to which the device instances in the domain belong. In the examples of this view, these classes have been identified by their "make" and "model number", which are unique across a class of device/component/system instance (such as "Fujitsu 2312" disk and "NCR 6411" printer). There may be many instances of an "NCR 6411" printer, each of which is identified by a unique serial number. All of these instances may be classified into the class of printers designated by the make ("NCR") and model number ("6411").

The relationship between the "instance" view and the "class" view is depicted in Figure 5. There are the entity sets DEVICE and DEVICES, which have as their extensions the set of device instances and the set of device classes to which the instances belong, respectively. These two entity sets are related through the MEMBER_OF classification abstraction relationship. The semantics of this relationship are as follows.

each DEVICE is_a MEMBER_OF exactly_one DEVICES class (1,1)
each instance of DEVICES is_a CLASS_OF at_least_zero and at_most_n instances of DEVICE (0,N)

Thus, each instance of the DEVICES entity set is a designated class for the instances of the DEVICE entity set. A class instance in DEVICES may have no associated device instances (i.e., elements in the DEVICE entity set) in the domain of discourse, or it may have many.

For both the instance and class level views of "devices", there is a corresponding IS-A hierarchy associated with it. The IS-A hierarchy of the DEVICE entity set is comprised of specializations of device instances into their respective sub-types, depending on whether they are printers, terminals, etc. The IS-A hierarchy of the DEVICES entity set indicates specializations of device classes into their respective sub-types. Both hierarchies are partitioned (indicated by the *XR integrity* constraint) according to their "device type". This "device type" is indicated by the **d_type** attribute, which is related to both instance and class hierarchies through a PROP_OF *attribution aggregation* relationship from the DEVICE_TYPE entity set.

The schema portion depicting the instance hierarchy is referred to as the *instance schema.* The schema portion depicting the class hierarchy is referred to as the *prototype schema*[14], because it represents the set of prototypical configurations and associations which are possible in the domain of discourse. In other words, the prototype schema captures in its extension the set of valid configurations which are possible between components and devices, disks and systems, devices and problem types, etc.

---

[14] This notion of "prototypes" is similar to that used in frame and semantic network representations [13] [6] [53] [61], in that it allows a *concept* of the domain to be defined, and allows the set of valid "expressions" of that concept to be stored in the extension of the schema. This is discussed further in [21].

## 5. Summary

This paper has presented an approach to constructing domain models as part of knowledge acquisition for a special type of knowledge-based system called an *organizational knowledge system*. This approach, known as Conceptual Abstraction, was presented as being useful in situations where some level of independence exists between the specification of the organizational knowledge and the underlying problem-solving methods that operate on that knowledge. Conceptual Abstraction was discussed in terms of its value in situations where there may be multiple problem-solving activities taking place simultaneously on a single or tightly-coupled knowledge store, or where the exact nature of the problem-solving tasks may not be known. The rationale for such an approach was shown to be mitigated by trends in industry and a confluence of developments between the AI and Database communities.

The focus of Conceptual Abstraction is on structuring domain descriptions of what is perceived about a domain by an "informant" in terms of a collection of abstraction constructs defined in a semantic modeling framework. The expressive power of these abstraction relations is augmented through additional language constructs that are used to capture semantic information about the domain being modeled. The modeling framework provides the basic vocabulary for expressing assertions about the domain of discourse (in terms of abstraction relations and various *cardinality*, *dependency*, and *interlink integrity* constraints), and provides a "visual" alternate to natural language approaches for structuring knowledge about the domain. This framework provides these abstraction primitives, and a set of guidelines that are used to construct domain models, as part of a larger knowledge acquisition methodology.

The intent is to provide a collection of modeling primitives that can be used to construct domain models in a structured manner; this structure has proved useful in helping the domain informant to express perceived domain semantics in precise and unambiguous terms. In addition, these primitives are proving their capability to support model-based reasoning strategies directed towards knowledge acquisition [25]. The use of Conceptual Abstraction in no way negates the need to use other elicitation techniques as part of the acquisition strategy; in fact, it seems that model-based reasoning about "artifactual" models under construction during acquisition may provide some automated guidance in the selection of appropriate elicitation mechanisms during the course of model acquisition.

This approach was developed as part of an environment to support knowledge based management systems comprised of multiple rule-based expert systems coupled to relational databases. Future work is being directed towards automating the domain model construction process, such that it becomes an integral part of the knowledge acquisition environment.

## 6. References

[1] Abrett, G., and M.H. Burstein, "The KREME Knowledge Editing Environment", in Boose, J.H., and B.R. Gaines (eds.), *Knowledge Acquisition Tools for Expert Systems*, Volume 2, Academic Press, 1988, pp. 1-24.

[2] Alexander, J.H, M.F. Freiling, S.J. Shulman, J.L. Staley, S. Rehfuss, and S.L. Messick, "Knowledge Level Engineering: Ontological Analysis, *Proceedings AAAI-86*, Philadelphia, PA, American Association for Artificial Intelligence, 1986, pp. 963-968.

[3] Alexander, J.H., M.J. Freiling, S.J. Shulman, S. Rehfuss, and S.L. Messick, "Ontological Analysis: An Ongoing Experiment", in Boose, J., and B. Gaines (eds.), *Knowledge Acquisition for Knowledge-Based Systems: Volume 2*, Academic Press, Inc., 1988, pp. 25-37.

[4] Anderson, J.R., *The Architecture of Cognition*, Harvard University Press, 1983.

[5] Brachman, R.J., "On the Epistemological Status of Semantic Networks", in Findler, N.V. (ed.), *Associative Networks: Representation and Use of Knowledge by Computers*, Academic Press, 1979, pp. 3-50.

[6] Brachman, R.J., "What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks", IEEE Computer, Oct. 1983, pp. 30-36.

[7] Bravos, A.G. "The Use of Entity-Relationship Models to Build Expert Systems", in Gordon, J.B. (ed.), *Proceedings of the National Communications Forum*, Vol. 41, 1987, pp. 1237-1241.

[8] Breuker, J., and B. Wielinga, "Use of Models in the Interpretation of Verbal Data", in Kidd, A. (ed.), *Knowledge Acquisition for Expert Systems: A Practical Handbook*, Plenum Press, Inc., 1987, pp. 17-44.

[9] Brodie, M., "Association: A Database Abstraction for Semantic Modelling", in Chen, P.P (ed.) *Entity-Relationship Approach to Information Modeling and Analysis*, ER Institute, 1981, pp. 583-608.

[10] Brodie, M.L, J. Mylopoulos, and J. Schmidt (eds.), *On Conceptual Modeling: Perspectives From Artificial Intelligence, Databases, and Programming Languages*, Springer-Verlag Publishers, Inc., 1984.

[11] Brodie, M.L, J. Mylopoulos (eds.), *On Knowledge Base Management Systems: Integrating Artificial Intelligence and Database Technologies*, Springer-Verlag Publishers, Inc., 1986.

[12] Bylander, T., and B. Chandrasekaran, " Generic Tasks for Knowledge-based Reasoning: The "Right" Level of Abstraction for Knowledge Acquisition", in Gaines, B., and J. Boose (eds.), *Knowledge Acquisition for Knowledge-Based Systems, Vol. 1*, Academic Press, Inc., 1988, pp. 65-77.

[13] Carbonell, J.G., "Default Reasoning and Inheritance Mechanisms on Type Hierarchies", in Brodie, M.L., and S.N. Ziles (eds.), *Proceedings of Workshop on Data Abstraction, Databases, and Conceptual Modeling*, Association for Computing Machinery, 1980.

[14] Chen, P.P, "The Entity-Relationship Model; Toward a Unified View of Data", *ACM Transactions on Database Systems*, Vol. 1, No. 1, pp. 9-36.

[15] Chen, P.P., "English Sentence Structure and Entity-Relationship Diagrams", *Information Sciences*, Vol. 29, 1983, pp. 127-150.

[16] Clancey, W.J., "Heuristic Classification", Stanford Knowledge Systems Laboratory, 1985.

[17] IEEE Computer, "Special Issue on Visual Programming", *IEEE Computer*, Vol. 18, No. 8, August 1985.

[18] Date, C.J., *Relational Database: Selected Writings*, Addison Wesley Publishers, Inc., 1986.

[19] Davis, J.P., and R.D. Bonnell, "Modeling Semantics with Concept Abstraction in the EARL Data Model", Fry, J.P. (ed.), *Proceedings 8th International Conference on Entity-Relationship Approach*, Springer-Verlag Publishers, Inc., 1989, pp. 102-117.

[20] Davis, J.P., and R.D. Bonnell, "Roles and Abstraction in the EARL Data Model", *USCMI Report #89-14*, Center for Machine Intelligence, University of South Carolina, 1989. Submitted for Publication, 1990.

[21] Davis, J.P., *The Knowledge Dictionary: A KBMS Architecture for the Many-to-many Coupling of Knowledge-based Systems to Databases*, PhD Thesis, University of South Carolina, 1989.

[22] Davis, J.P., and R.D. Bonnell, "The Knowledge Dictionary: A Repository-Based KBMS Framework for Organizational Knowledge Systems", *USCMI Report #89-30*, Center for Machine Intelligence, University of South Carolina, 1989. Submitted for Publication, 1990.

[23] Davis, J.P., and R.D. Bonnell, "Modeling Semantic Constraints with Logic in the EARL Data Model", *Proceedings Fifth International Conference on Data Engineering*, IEEE Press, 1989, pp. 226-233.

[24] Davis, J.P., and R.D. Bonnell, "Concept Abstraction and the Knowledge Dictionary: A Framework for Acquisition and Maintenance of Knowledge Schemata", in *Proceedings IJCAI-89 Workshop on Knowledge Acquisition: Practical Tools and Techniques*, Detroit, MI, August 23, 1989. Extended paper submitted for Publication, 1990.

[25] Davis, J.P., "Abstraction-Based Reasoning: Somewhere Between the Shallow and the Deep", *HRI Research Report #90-3-1*, Horizon Research Inc., 1990. In preparation.

[26] Debanne, X., "A Knowledge Model for Expert Systems, Based on a Data Model", in Kerschberg, L. (ed.), *Proceedings of the 1st International Workshop on Expert Database Systems*, Institute of Information Management, Technology, and Policy, University of South Carolina, 1984, pp. 656-664.

[27] Drucker, P.F., "The Coming of the New Organization", *Harvard Business Review*, Jan-Feb. 1988, pp. 45-53.

[28] Harandi, M.T., and R. Lange, "Model-Based Knowledge Acquisition", in Adeli, M. (ed.), *Knowledge Engineering: Volume 1, Fundemantals*, McGraw-Hill, Inc., 1990, pp. 103, 129.

[29] Hayes, P.J., "On Semantic Nets, Frames, and Associations", *Proceedings IJCAI-79*, Morgan-Kaufmann Publishers, pp. 99-107.

[30] Hull, R., and R. King, "Semantic Database Modeling: Survey, Applications, and Research Issues", *ACM Computing Surveys*, Vol. 19, No. 3, September 1987, pp. 201-260.

[31] Huff, K.E., and V.R. Lesser, "The Grapple Plan Formalism", *COINS Technical Report #87-08*, University of Massachusetts at Amherst, April 1987.

[32] Huhns, M.N., *Distributed Artificial Intelligence*, Morgan Kaufmann Publishers, Inc., 1987.

[33] Jackendoff, R.S., *Semantics and Cognition*, MIT Press, 1983.

[34] Johnson, N.E., "Mediating Representation in Knowledge Elicitation", in Diaper, D. (ed.), *Knowledge Elicitation: Principles, Techniques, and Applications*, Ellis Horwood, Ltd., 1989, pp. 179-194.

[35] Kerschberg, L. (ed.), *Expert Database Systems*, Benjamin Cummings Publishing Co., 1987.

[36] Kumara, S.R.T., and R.L. Kashyap, "Knowledge Representation in Expert Systems via Entity-Relationships and its Applications", *Proceedings 1st Conference on Applications in AI*, IEEE Press, 1985, pp. 495-500.

[37] Lee, E.T., P. Chu, and C.P. Wu, "Applications of Entity-Relationship Model to Picture Description", *Robotica*, Vol. 5, pp. 231-234.

[38] Lenat, D., M. Prakash, and M. Shepherd, "Cyc: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks", *The AI Magazine*, Vol. 6, No. 4, 1986, pp. 65-85.

[39] Linster, M., "Towards a Second Generation Knowledge Acquisition Tool", *Knowledge Acquisition*, Vol. 1, 1989, pp. 163-183.

[40] Marcus, S. (ed.), *Automating Knowledge Acquisition for Expert Systems*, Kluwer Academic Publishers, 1988.

[41] McDermott, J., "Preliminary Steps Toward a Taxomony of Problem-solving Methods", in Marcus, S. (ed.), *Automating Knowledge Acquisition for Expert Systems*, Kluwer Academic Publishers, 1988, pp. 225-256.

[42] Morik, K., "Acquiring Domain Models", in Boose, J., and B. Gaines (eds.), *Knowledge Acquisition for Knowledge-Based Systems, Vol. 2*, Academic Press, Inc. 1989, pp. 245-256.

[43] Musen, M.A., L.M. Fagin, E.H. Shortliffe, Graphical Specification of Procedural Knowledge for an Expert System", *Proceedings 1986 Workshop of Visual Languages*, IEEE Computer Society, 1986, pp. 167-178.

[44] Musen, M.A., *Automated Generation of Model-Based Knowledge Acquisition Tools*, Pitman Publishers, Ltd., 1989.

[45] Mylopoulos, J., "A Perspective for Research on Conceptual Modelling", in Brodie, M.L., and S.N. Zilles, *Proceedings on Workshop on Data Abstraction, Databases, and Conceptual Modeling*, Association for Computing Machinery, 1980, pp. 167-170.

[46] Mylopoulos, J., and H.J. Levesque, "An Overview of Knowledge Representation", in M.L. Brodie, et al (eds.), *On Conceptual Modeling: Perspectives from Artificial Intelligence, Databases, and Programming Languages*, Springer-Verlag, 1984, pp. 3-17.

[47] Regoczei, S., and E.P.O. Plantinga, "Creating the Domain of Discourse: Ontology and Inventory", in Boose, J., and B. Gaines (eds.), *Knowledge Acquisition for Knowledge-Based Systems, Vol. 2*, Academic Press, Inc., 1989, pp. 293-308.

[48] Regoczei, S., and G. Hirst, Knowledge Acquisition as Knowledge Explication by Conceptual Analysis, *Technical Report CSRI-205*, Computer Systems Research Institute, University of Toronto, January, 1988.

[49] Schubert, L.K., M.A. Papalaskaris, and J. Taugher, "Determining Type, Part, Color, and Time Relationships", *IEEE Computer*, Oct. 1983, pp. 53-60.

[50] Sernadas, A., and C. Sernadas, "The Use of E-R Abstractions for Knowledge Representation", *Proceedings 4th International Conference on Entity-Relationship Approach*, IEEE Press, 1985, pp. 224-231.

[51] Smith, J.M., and D.C.P. Smith, "Database Abstractions: Generalization and Aggregation", *ACM Transactions on Database Systems*, Vol. 2, pp 105-133.

[52] Soloway, E., J. Bachant, and K. Jensen, "Assessing the Maintainability of XCON-in-RIME: Coping with the Problems of a VERY Large Rule-Base", in *Proceedings AAAI-87*, Seattle, WA, American Association for Artificial Intelligence, 1987, pp. 824-829.

[53] Sowa, J.F., *Conceptual Structures: Information processing in Mind and Machine*, Addison Wesley Co., 1984.

[54] Swigger, K.M., and R.P. Brazile, "An Architecture for an Intelligent Tutoring System Based on an Entity-Relationship Data Model", *Intelligent Tutoring Systems*, Montreal, June 1988, pp. 258-264.

[55] Tsichritzis, D.C., and F.H. Lochovsky, *Data Models*, Prentice-Hall, Inc., 1982.

[56] Vicens, L.J., R.E. Markland, K.R. Karwan, and J.P. Davis, "An Integrated Methodology for Knowledge Based Systems Development: The TACTICS Experience", in Karwan, K.R., and J.R. Sweigart (eds.), *Proceedings 3rd International Conference on Expert Systems and the Leading Edge in Production and Operations Management*, College of Business Administration, University of South Carolina, 1989, pp. 109-124.

[57] *IEEE Workshop on Visual Languages*, University of Pittsburg, Oct. 10-12, 1988, IEEE Press, 1988.

[58] Whorf, B., *Language, Thought, and Reality*, MIT Press, 1956.

[59] Williams, G., and N. Iscoe, *Proceedings of Workshop on Domain Modeling for Software Engineering*, New Orleans, LA, 1989.

[60] Winston, M.E., R. Chaffin, and D. Herrmann, " A Taxonomy of Part-Whole Relations", *Cognitive Science*, Vol. 11, 1987, pp. 417-444.

[61] Brachman, R.J., "'I Lied About the Trees' Or, Defaults and Definitions in Knowledge Representation", *The AI Magazine*, Fall 1985, pp. 80-93.

[62] Brodie, M.L., "Future Intelligent Information Systems: AI and Database Technologies Working Together", in Mylopoulos, J., and M.L. Brodie (eds.), *Readings in Artificial Intelligence and Databases*, Morgan Kaufmann Publishers, Inc., 1988.

# Integration of Knowledge from Different Knowledge Acquisition Tools

Jorun Eggen
ELAB-RUNIT, SINTEF, N-7034 Trondheim, Norway
E-mail: jorun.eggen@elab-runit.sintef.no

Astrid Marie Lundteigen[1]
Norsk Forsvarsteknologi A/S, P.O.Box 1003, N-3601 Kongsberg, Norway

Mette Mehus
EB Technology, P.O.Box 90, N-1361 Billingstadsletta, Norway

## Abstract

This paper presents a computational model of integration of new knowledge into an existing knowledge base. Our model is an extension of Murray's model[8] and describes the integration process in four steps:
- Ordering the knowledge pieces in the new knowledge into an integration order, recognition of relevant knowledge in the existing knowledge base, elaboration of the consequences of adding the new knowledge and adaptation of the new knowledge into the existing knowledge base.

The paper also presents a decomposition of the integration process into several subtasks together with a set of functions required to support the knowledge engineer in each subtask. Different representation constructs have to be checked in different ways, and this is reflected in the subtask decomposition and the identified functions. Knowledge integration is viewed as a modelling task, carried out in interaction between the KEW and the user.

The work is a part of the ACKnowledge project (Esprit 2576) where the Knowledge Engineering Workbench (KEW) will have extensive support for integration of knowledge acquired by the various knowledge acquisition tools in KEW.

[1]Astrid and Mette participated in this work while they were doing their Master Thesis [6] at NTH - The Norwegian Institute of Technology.

# 1 Introduction

*ACKnowledge*[2] (Esprit project 2576) [3] is an acronym for ACquisition of Knowledge [7]. The objective of the project is to improve the efficiency of the knowledge acquisition process in all phases of development of a knowledge based system. This will be realized in two ways:

- The development of methods and techniques for integrating several knowledge acquisition techniques.
- The design and implementation of an environment which integrates a number of existing automated and non-automated techniques; the *Knowledge Engineering Workbench - KEW.*

The KEW will assist the knowledge engineer in his tasks, and partially provide automation of some of these tasks. The key constituents of the workbench are three kinds of knowledge acquisition tools: machine learning tools (similarity based and case based learning), modelling tools (protocol analyzer, knowledge structuring tool and knowledge base editors) and knowledge elicitation tools (card sort, laddering tool and repertory grid).

The approach taken in ACKnowledge to resolve the knowledge acquisition problem is to integrate into a coherent system the tools that support different parts of knowledge acquisition. The integration of tools is realized through:

**A common user interface** to the KEW and thus each of the tools.

**A common information repository** which contains all the knowledge and relevant information that is produced during the knowledge engineering process

**A set of functions for transformation and integration of knowledge** from the various tools into the Knowledge Base.

**An active module for advice and guidance** which follows the user's operations in KEW and because it has knowledge about the knowledge acquisition process in general, and KEW in particular, it can give advice to the user on what to do next, based on the current situation in the knowledge base and the history of the current project.

[2]SINTEF's work in ACKnowledge is funded by SINTEF and NTNF - The Norwegian Council for Technical and Industrial Research.

[3]The ACKnowledge project has seven European consortium members. The partners are: Cap Sesa Innovation (Prime Contractor), Marconi Command and Control Systems – GEC–Marconi Research Center (Sub-Contractor), Telefonica - U.P. Madrid (Sub-Contractor), Computas Expert Systems - Veritas Research (Sub-Contractor), University of Amsterdam, SINTEF, University of Nottingham.

The purpose of this paper is to outline a computational framework for the integration functions in KEW; the emphasise of the paper is therefore on the KEW and not on the method development work within ACKnowledge.

Abbreviations in this paper are limited to three: *KA - knowledge acquistion*, *KB - knowledge base* and *KEW - knowledge engineering workbench.*

# 2 The Knowledge Engineering Workbench - KEW

Further details can be found in [2] which gives a conceptualisation of the KEW. Here four major classes of functions in KEW are described:

**Application Building Functions,** which comprises all the functions directly related to the knowledge acquisition process. This includes knowledge acquisition tools together with functions for transformation and integration of knowledge, for direct editing of the knowledge base, for evaluation of the current status of the knowledge base, for planning of the subsequent KA-process and for providing advice and guidance to the user based on the current state of the KA-process.

**Project Management Functions,** which are the functions for maintaining project management information.

**Browse and document functions,** which comprises the functions for inspection of the information and knowledge stored in KEW and for providing documentation of it.

**Target Knowledge Base Generation Functions,** which are the functions supporting the generation of the target KB from the KB resulting from the KA-process using KEW.

The ultimate goal of the knowledge engineering process is a knowledge based system running in the target environment solving the application tasks with a satisfactory quality. The final output of the KEW will be an *Integrated Knowledge Base - IKB* which will have the knowledge and problem solving quality of the Target KB, but which is not runable in the target environment and maybe lacks the required efficiency. The KEW will contain functions supporting the final generation of the Target KB from the Integrated KB. This generation will not take place until the Integrated KB satisfies the project-defined quality criteria. Thus the Integrated KB is the central goal of the user's work with KEW.

To achieve this goal, the various tools in KEW are used to produce several types and versions of *Intermediate Information Structures - IISes* - like sets of cases, repertory grids, think aloud protocols, sets of rules from induction tools etc. These structures will

be represented in heterogeneous representation languages[4], and they will to some extent be acquired independently of each other due to the iterative and incremental nature of the KA-process. Thus, in order to make use of these structures to construct the Integrated KB, each of them has to be **assimilated** into the Integrated KB. This can be done in two ways:

1. The manual way:
   The knowledge engineer interprets the Intermediate Information Structures and by means of direct editing of the Integrated KB he creates knowledge structures like concepts and rules, and thus integrates the knowledge content of the Intermediate Information Structure into the Integrated KB.

2. The KEW supported way:
   The Intermediate Information Structures are transformed into the same representation language as the Integrated KB, and stored into a *Temporary Knowledge Base*. This Temporary KB may be integrated into the Integrated KB through interaction between the KEW and the knowledge engineer.

The former alternative comes from the fact that some of the Intermediate Information Structures have a form which is not interpretable by the KEW. E.g. think aloud protocols which are "marked" with hypertext links: there is no way the transformation function can pick out the relevant knowledge and transform it to the core representation language[5]. Thus the user has to interpret the Intermediate Information Structure and manipulate the Integrated KB directly to integrate the knowledge[6].

The latter alternative is applicable for the Intermediate Information Structures that have a more structured output, like a set of rules or a repertory grid.

Fig.1 illustrates the two alternatives of the assimilation process[7]. The figure also shows the possibility of transformation from the Integrated KB to an Intermediate Information Structure, if the Intermediate Information Structure needs input from the existing knowledge, and it shows the possibility of transformations between Intermediate Information Structures. The upper area has heterogeneous representation languages, while the lower area has one homogeneous representation language.

[4]One of the goals of ACKnowledge is to enable integration of *existing* KA-tools in KEW; thus the resulting knowledge structures from these tools will be in different knowledge representation languages.

[5]The representation language used in the Temporary and the Integrated KB.

[6]A third alternative would be to build an interpretable Intermediate Information Structure from this one, and then proceed with transformation and integration.

[7]The figure is based on several figures proposed in ACKnowledge.

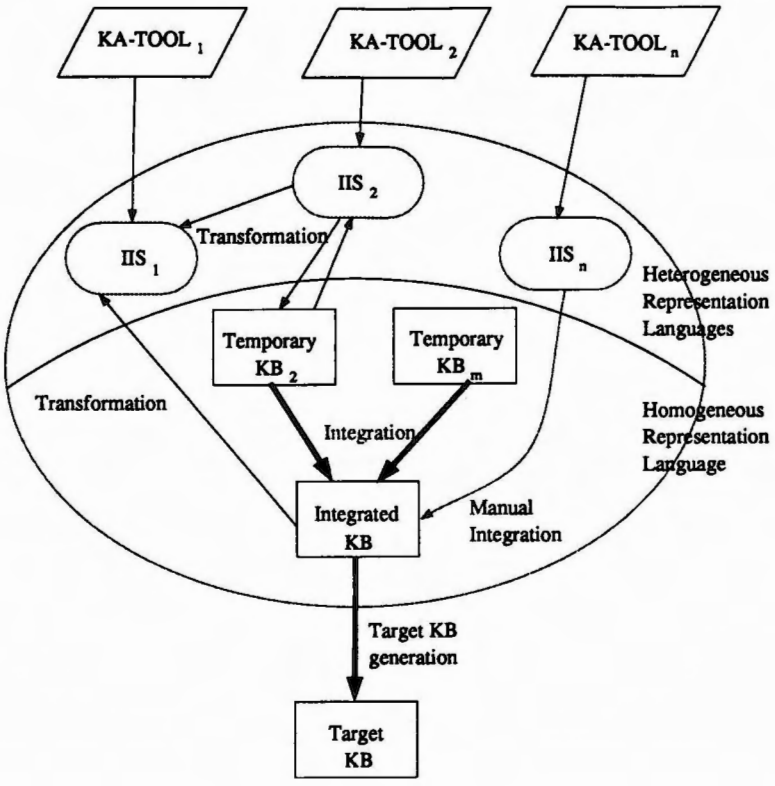



Figure 1: The assimilation process and the knowledge bases involved

The constituents of the assimilation functions are:

**Transformation**

- converts from one formalism to another
- may be supported by domain specific background knowledge (knowledge from user, knowledge of domain, knowledge about representation etc.)
- fully automatic or semi-automatic

**Integration**

- input and output are in the same formalism
- highly domain knowledge intensive
- maintain (or resolve) concistency (local[8])
- semi-automatic

Studies of transformation within ACKnowledge (see [4]) discuss knowledge supported transformation, but prototyping experiments in the ACKnowledge project indicate that the transformation functions in KEW will be mainly syntax translation. This reduces the complexity and thus the transformation functions may be fully automated. The hard problem is to decide how a certain construct in one representation language should be represented in the target language. This is done once and for all when the transformation algorithm between languages A and B is written.

The consequences are that even more of the success in the approach we have taken in ACKnowledge relies on the integration functions.

# 3 Two Examples

To illustrate some of the problems that occur in knowledge integration we present two examples from our pilot study on integration of knowledge from KEW-0[9].In the pilot study we produced two Temporary KBs by using TACKTIX (a repertory grid tool [1]) and INSTIL (a machine learning tool based on induction [5]) to acquire knowledge. Then we hand-simulated the integration of this knowledge into the existing

[8]Global inconsistencies may be acceptable within a domain. Like the two theories about light - the particle theory and the wave theory - they exist in parallel and are used for different purposes. Local consistency means that each of these theories must be internally consistent.

[9]KEW-0 is the first prototype of the KEW, where some KA-tools made available by the ACKnowledge partners were put together in one system providing common access to all tools, but with no common knowledge base to transfer the knowledge into. KEW-0 is operational and some experimentation on transformation and integration has been carried out in the prototype.

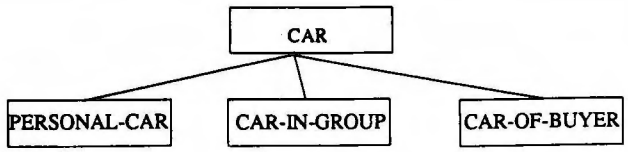



Figure 2: The frame hierarchy in the Integrated KB.

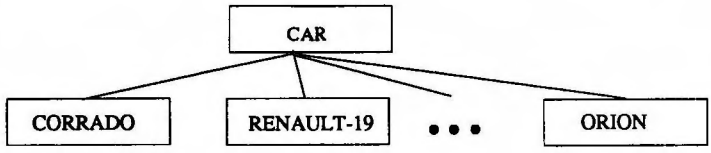



Figure 3: TKB1 - The frame hierarchy to be integrated.

KB in METABUILD (a modelling tool providing a structure editor and a rule editor and a representation language[9],[3]. In this context METABUILD was used to simulate an Integrated KB.) This pilot study revealed a lot of the problems the integration functions have to deal with, and our two examples here are scenarios to illustrate the problems and possible solutions.

The first example deals with integration of a frame structure into the Integrated KB, while the second example deals with integration of rules. In the examples Integration functions are referred to in capital letters. These functions will be described in chapter 5.

**First Example:**

In our example we have a small Integrated KB as fig.2 shows. The use of TACKTIX has given us a Temporary KB - TKB1 - which contains the frame hierarchy shown in fig.3. The following will illustrate how the KEW might help us to integrate TKB1 into the Integrated KB.

One of the tasks of the integration function is to compare the new piece of knowledge with the Integrated KB to possibly identify a corresponding part in the Integrated KB. In this example a corresponding part is found, because the top frame of the TKB1 has the same name as a frame existing in the Integrated KB; the car frame. Then each of the frames in the TKB1 is compared with the frames in the Integrated

KB by applying the function FIND-CORRESPONDING-FRAME. When comparing the two CAR-frames, a dialogue between the KEW and the user might look like this:

*KEW:* There exists a frame called PERSONAL-CAR in the IKB which seems similar to the CAR frame from the TKB1. PERSONAL-CAR from IKB and CAR from TKB1 probably represent the same concept. Please notice the following:

- There does exist a CAR frame in the IKB. This frame is hardly the similar frame as the CAR frame from the TKB1.
- There does not exist any frame in the TKB1 with the name PERSONAL-CAR.

Do you want to look at the content of the frames?

*KE:* No.

*KEW:* Do you want to look at the explanation related to a frame?

*KE:* Yes.

*KEW:* Which frame explanation do you want to see?

*KE:* I want to see the explanation to the PERSONAL-CAR frame.

*KEW:* (Showing the explanation belonging to the PERSONAL-CAR frame.)

You have the following choices when determining a possible similar frame to the CAR frame from the TKB1:

1. The PERSONAL-CAR frame and the CAR frame are similar.
2. The CAR frame (from IKB) and the CAR frame (from TKB1) are similar.
3. Another frame is similar to the TKB1 frame.
4. There does not exist a corresponding frame.

Select one of the possible choices:

*KE:* The PERSONAL-CAR frame and the CAR frame correspond.

Finally, the knowledge engineer has to decide whether to keep the PERSONAL-CAR or the CAR name, or to introduce a quite new name.

**Second Example:**

From the same car-scenario we present an example of integrating rules about types of customers. INSTIL has been used to induce a set of rules that is to be integrated in the Integrated KB. Here we have come to the point of checking superfluity and conflicts. The KEW presents the rules and the result of the checking. A possible output is shown in fig.4.

KEW does not discover any superfluities for the rules 2 or 4. In rule number 11, however, KEW concludes that some parts of the premises are superfluous.

Rule number 11 states that the customer has to be a yap or a spoilt-son, which implies that rules number 2 and 4 have to be TRUE, assuming that there are no other rules that can conclude "yap" or "spoilt-son". As a consequence, if rule number 11 should be TRUE, the customer has to have income=3 and sex=male. The premises concerning the sex and income in rule number 11 will therefore be superfluous (rule number 11 states that income can have the value 2, but, we will never obtain the right type of customer if the income has value 2). The knowledge engineer decides to change rule number 11 to be like the following:

No 11: (type OF customer = yap ∨ type OF customer = spoilt-son) → recomm-car-type OF customer = sports-car

We then continue checking whether any conflicts were present. After removing superfluous components, we deduce all the consequences from incorporating the new rules. From the deduced consequences, possible conflicts are discovered and solved. For performing these tasks, the function CHECK-IF-CONFLICTS will be applied for each rule.

When KEW deduces consequences from including rule number 4, we get a situation as illustrated in figure 4. In relation to rule number 4 a new rule is deduced on the basis of premises making both rule number 2 and 4 TRUE. The new rule's conclusions constitute a set of possible conclusions. A possible presentation of the results for the knowledge engineer could be like this:

*KEW:* From rule number 2 and 4, a new rule is deduced. This rule may be interpreted in either of the following ways:

1. The new rule expresses a legal relation between the premises and the conclusions. It is possible to let a customer be a yap and a spoilt-son concurrently.
2. One or both of the rules are not specified in enough detail. It is possible to change or add premises in order to obtain rules which do not have overlapping premises.
3. There may exist a conflict. It may be impossible that a customer can be of both type yap and type spoilt-son. One or both of the rules may be wrong.

No 2: income OF customer = 3
age OF customer < 35
sex OF customer = male
→ type OF customer = yap

⇝ *Superfluities:*
No superfluities

⇝ *Conflicts:*
Not in conflict with IKB

No 4: age OF customer < 30
income OF customer = 3
sex OF customer = male
→ type OF customer = spoilt-son

⇝ *Overlap:*
2 and 4: income OF customer = 3
age OF customer < 30
sex OF customer = male
⇒ type OF customer = (yap, spoilt-son)

•
•
•

No 11: sex OF customer = male
(income OF customer = 2
∨ income OF customer = 3)
(type OF customer = yap
∨ type OF customer = spoilt-son)
→ recomm-car-type OF customer = sports-car

⇝ *Superfluities:*
(type OF customer = yap
∨ type OF customer = spoilt-son)
⇒ sex OF customer = male is always true
⇒ income OF customer = 3 is always true
⇒ income OF customer = 2 is never tested

Figure 4: How to discover superfluities and conflicts.

The knowledge engineer decides to conclude that the new rule expresses a legal relationship.

# 4 A Computational Model of Integration

Integration of knowledge is a process which combines knowledge from different knowledge acquisition tools which may originate from different sources. Different textbooks and different experts obviously would use slightly different terminology, have different rules-of-thumb and even slightly different theories. If the KA-process spans a significant time period, the same problems with variations in terminology etc. may occur with a single expert as well. This of course introduces serious problems in adapting new knowledge into the existing KB. The tools may have acquired the same or different knowledge, implying that knowledge in one tool may extend or refine the knowledge from another tool. Even though the same knowledge is represented in several tools, the semantics of the different tools may be different, thereby leading to possible misinterpretations. The integration must take care of the semantics in the knowledge to be sure that the knowledge piece in the Temporary KB and the Integrated KB "speaks the same language". I.e. that the meanings of a given term in the two KBs are actually equal. The knowledge in the Integrated KB must be used for this purpose.

The main objective of the integration process is to identify semantical similarities and detect possible inconsistencies and incorrect knowledge. The integration process should also reveal whether any knowledge is missing and should be supplied. This will be an interactive process, and the knowledge engineer should have a lot of support for:

- checking of constraints
- checking of completeness
- checking of correctness
- checking of consistency

The notion of integration of knowledge is a new issue in the knowledge engineering field, and very few publications exist. Murray's work [8] where he regards integration of knowledge as a knowledge rich type of machine learning, is the only closely related work we have found so far.

Murray's definition of integration of knowledge is:

*Knowledge integration is the incorporation of new information into existing knowledge. It requires elaborating new information and resolving inconsistencies with existing knowledge.*

He summarizes the knowledge integration task as follows:

Given:

1) *Representation Language:* A language to describe beliefs (e.g. rules and facts).
2) *Knowledge Base:* A set of existing beliefs.
3) *Training:* A set of new beliefs which are to be integrated into the existing knowledge base.
4) *Admissibility Criteria:* A predicate over knowledge bases, specifying criteria that beliefs of a revised knowledge base must satisfy (e.g. consistency requirements).

Determine:

*A Revised Knowledge Base:* A set of beliefs drawn from the training, the knowledge base and their extensions, that satisfies the admissibility criteria.

In KEW we find parts corresponding to this model:

1. The representation language is the core knowledge representation language in which the Integrated KB and the Temporary KBs are represented.
2. The knowledge base is the Integrated KB.
3. The training set is one or more Temporary KBs.
4. The admissibility criteria are the consistency requirements the integration functions strive to maintain, e.g. concepts with the same relations and same attributes are equal, mutually exclusive predicates cannot appear as premises in the same rule. The richness of these criteria depends on the expressiveness of the representation language.
5. The revised knowledge base is the updated Integrated KB.

Murray views the integration of knowledge as a knowledge rich type of machine learning. In the ACKnowledge environment the integration of knowledge can be viewed as a **modelling activity**, where the knowledge engineer works to refine the Integrated KB - the model of the application. The integration function in KEW is a tool for supporting this activity.

What is most important in Murray's approach seen in relation to the ACKnowledge project, is the description of the knowledge integration process. He proposes the REACT[10] framework which divides the process into three steps - recognition, elaboration and adaptation - and different requirements are related to each stage. This

[10]REACT is an abbreviation of the *Recognition, Elaboration, Adaptation Computation Theory.*

computational model provides a description of the different tasks involved in knowledge integration.

Our pilot study uncovered a weakness in Murray's framework: With what knowledge structure in the training set should the recognition start? In KEW the training set will normally be a Temporary KB consisting of a number of knowledge pieces like concepts, relations, rules etc. Only rarely will the training set be a single concept or a single rule.

Thus in our approach to knowledge integration we have extended Murray's model with a first step to decide the **order** of integration. This step is added to handle situations when there is more than one Temporary KB to integrate, and more than one "piece" of knowledge within a Temporary KB. Then the integration function has to decide which piece to integrate first. The order seems important for the efficiency of the integration process; Whether it is important to the quality of the resulting knowledge base remains an open question.

This gives us a computational model with the following four steps:

- **Ordering - a search for an integration order of the knowledge pieces in the training set**

  The learner[11] selects the next knowledge piece[12] to be integrated.

- **Recognition - a search for relevant background knowledge**

  The learner selects relevant knowledge structures to guide the interpretation of new information provided as training examples.

- **Elaboration - a search for consequences**

  The information content of the training is expanded as expectations and inferences provided by the retrieved knowledge structures are applied.

- **Adaptation - a search for accommodation**

  The learner's knowledge is modified to accommodate the elaborated information.

# 5 A Task Decomposition of Integration

We will emphasize that the integration process cannot be fully automated. The integration of knowledge concerns the meaning of the knowledge, which is very difficult for an automatic integration system to fully understand. The integration process must

[11] In our approach the learner is the KEW proposing and the user responding to these propositions.

[12] A knowledge piece can be either a Temporary KB, a part of a Temporary KB such as the structure part of the resulting KB from INSTIL, or a single rule or frame.

therefore be implemented as an interactive system in order to let the knowledge engineer and the supporting integration functions work together. The integration functions we have identified are therefore meant as a support to the knowledge engineer. The knowledge engineer always has the possibility to overrule the suggestions made by the integration system.

The integration of knowledge is a large and important task. As a consequence, the integration problem has to be divided into several smaller tasks each being thoroughly defined. In the following, we give a survey of the main integration tasks we have identified. We have compared each task with the four integration steps defined in our computational model.

1. **Deciding integration order of knowledge parts:**

   When integrating knowledge into the Integrated KB, the knowledge may come from different knowledge acquisition tools. Each tool may have one or several knowledge bases. The first problem discovered is to determine which knowledge base to integrate first and whether to integrate a whole knowledge base at the same time or not. In our pilot-study, most of the knowledge bases were divided into two *parts*; a structure part and a rule part. We assume that the integration process will integrate one of these knowledge parts at a time.

   We therefore recommend that the integration system should offer a function recommending which knowledge part to integrate next. The resulting order should be presented for the knowledge engineer who then can decide to follow the system recommendation or choose his own ordering. This task corresponds to the *ordering* step in the computational model.

   A structure knowledge part and a rule knowledge part may be very different. We therefore need some functions constructed especially for structures and some functions constructed especially for rules. As a consequence, the tasks identified for integrating structures and integrating rules are described separately.

2. **Main tasks when integrating a structure knowledge part:**

   We will now describe the main integration tasks which have to be solved when integrating a structure knowledge part.

   (a) **Deciding integration order within a structure part:**

   A structure knowledge part consists of one or several frames organized in one or several independent hierarchies[13]. The order will be dependent of the

[13]By *independent hierarchies* we mean hierarchies with no relations between them. There may, though, be a semantical relation, but this is difficult for the integration system to detect and must be identified by the knowledge engineer.

existing knowledge in the Integrated KB. The problem is to find the order in which we should integrate the different hierarchies, and which integration order to use for the frames within the same hierarchy. An integration function suggesting an order for integrating the frames should be offered. The knowledge engineer can then choose to follow the system's recommendations or he can decide his own integration order for the frames.

This task corresponds to the *ordering* step in the model.

(b) **Identifying corresponding part of the Integrated KB:**

The frame to be integrated may represent a concept allready existent in the Integrated KB. It may have a different name and it may have different attributes and relations, but from the knowledge point of view it is the same concept. The KEW has to search for a corresponding frame first, and if no corresponding frame is found, a possible parent for the frame has to be found.

The task described here corresponds to the *recognition* step.

(c) **Solving conflicts and including a frame:**

We have now found the position in the Integrated KB where we want to include the frame. If we have concluded that two frames are corresponding, they do not need to be exactly similar. We therefore have to test if similar terms in the two frames have different meanings, we have to test if conflicts will occur due to inheritance or due to the different use of attribute values etc. The integration system should have an interactive function testing and solving all these possible situations. The frame should then be ready for inclusion in the Integrated KB.

This function will fit both the *elaboration* and the *adaptation* phases of the computational model. The conflict solving corresponds to the elaboration and the inclusion corresponds the adaptation.

(d) **Checking and improving completeness**

After a frame is included in Integrated KB, it may lack attributes, facets and values. There may also be a need for identifying possible lacking relations. A function identifying possible missing components should be offered. The integration system should support the knowledge engineer in detecting possibly incomplete knowledge.

This function is part of the *adaptation* task.

3. **Main tasks when integrating a rule knowledge part:**

We will now introduce some main integration tasks which must be solved when integrating a rule knowledge part. Some tasks are almost similar to the tasks introduced when integrating a structure knowledge part.

(a) **Deciding integration order of rules:**

This function is very similar to the function recommending an integration order of frames from a structure knowledge part. When integrating a rule base, we will normally integrate one rule at a time. The integration system should offer a task recommending an integration order of the rules. This is meant to be a help to the knowledge engineer who can follow the recommendation or decide his own integration order.

This task corresponds to the *ordering* step.

(b) **Recognizing the terms of a rule**

Every term used in a rule must be defined as a frame, an attribute, a facet, a relation or a legal value for one of these. When a rule is to be integrated we must ensure that all terms are defined, otherwise they must be defined to enable checking of similarity and conflicts. Here we have the same problem as when recognizing a frame - there may be problems with terminology.

This task correspond to the *recognition* step.

(c) **Checking superfluity:**

We now have a rule to be integrated in Integrated KB. It may be that this rule, or parts of the rule, already exist in the Integrated KB. As a consequence, there may be rules or parts of rules which are superfluous. The integration mechanism should therefore define a function checking whether a new rule leads to superfluities or not. If some superfluities are discovered, the function should remove them.

If the new rule we integrate is superfluous, we do not have to perform more functions related to this rule. If the function checking superfluity concludes that some parts of the rule are superfluous, we have to change either the rule or some parts of the Integrated KB in order to keep the Integrated KB free from redundancy.

The integration task described here corresponds to the *elaboration* step.

(d) **Deducing consequences and resolving conflicts:**

A rule may influence many components of the Integrated KB. All the consequences derived from integrating a new rule should be evaluated. The rule may modify some attribute values in some frames, the rule may have a conclusion which is a premise in another rule (integrating the first rule will therefore influence the second rule) etc. We often get some conflicts when examining the consequences of a rule. These conflicts must be solved before integrating the rule.

This task will fit the *elaboration* step.

(e) **Deciding position in the Integrated KB:**
When a rule is to be integrated into the Integrated KB, there should be an integration function recommending a suitable position for the new rule in the Integrated KB.
This task is a part of the *adaptation* phase.

(f) **Including a rule:**
The final task in the process of integrating a rule is to include the rule in the Integrated KB's rule base. We therefore need a function including a rule.
This last task corresponds the *adaptation* step.

[6] describes more detailed functions for each of the tasks described above, and table 1 (adapted from [6]) visualizes the relations between the main integration subtasks and the required functions. The table thus gives a survey of all the main functions which have to be performed when integrating a Temporary KB.

# 6 Challenging Issues

Knowledge integration is not a straightforward task, and we like to present some of the challenges that have to be worked through:

- Importance of order:
  Ordering is a necessary function, but how important is it to decide an optimal order? Does the order influence the quality of the KB? How important is the order to get an efficient integration process? Can recommendations on integration order be useful for recommending how a set of KA-tools can be used most efficiently?

- Integration based on the application knowledge:
  As the Integrated KB is improved from nothing into a comprehensive model of the application, the integration functions should use this knowledge as a basis in their checking, especially in elaborating the consequences of the new knowledge. How can this be achieved?

- Integration of knowledge from tools that have different focus:
  Repertory grids, for instance, have constructs corresponding to the attributes of a concept, and these constructs are rated on a given scale for each element in the grid; e.g. The element "Mercedes" has a construct "price"=5. In the Integrated KB one would more likely represent the price of a car as the actual price, say 400 000 NOK. Representation of uncertainty is a similar example. How are such differences in the representations to be handled by the integration functions?

Table 1: Hierarchical illustration of the main integration functions.

| Integration functions for knowledge parts: |
|---|
| *Decide the integration order of knowledge parts*<br>• DECIDE-ORDER-OF-TKB |
| *Integration of structures*<br>**Deciding the integration order within a structure part**<br>• DECIDE-ORDER-OF-FRAMES<br><br>**Deciding position in the IKB**<br>• FIND-CORRESPONDING-FRAME<br>- TEST-SIMILARITY<br><br>• FIND-POSSIBLE-PARENT<br>**Resolving conflicts and including a frame**<br>• INCLUDE-FRAME<br>- CHOOSE-TERM<br>- ATTRIBUTE<br>⋆ TEST-SIMILARITY<br>⋆ CHOOSE-TERM<br>⋆ INTEGRATE-FACET<br>◦ TEST-SIMILARITY<br>◦ CHOOSE-TERM<br>◦ INTEGRATE-VALUE<br>◇ TEST-VALIDITY<br>- INTEGRATE-RELATION<br><br>**Checking and improving completeness**<br>• CHECK-AND-IMPROVE-COMPLETENESS |
| *Integration of rules*<br>**Deciding the integration order of rules**<br>• DECIDE-ORDER-OF-RULES<br><br>**Recognition of terms**<br>• RECOGINZE-TERMS<br><br>**Checking superfluity**<br>• CHECK-IF-SUPERFLUOUS<br><br>**Deducing consequences and resolving conflicts**<br>• CHECK-IF-CONFLICTS<br>- CHECK-IF-SUPERFLUOUS<br><br>**Deciding position in the IKB**<br>• DECIDE-RULE-POSITION<br><br>**Including a rule**<br>• INCLUDE-RULE |

- Integration of changeable knowledge:
  Due to the iterative nature of the KA-process, the same repertory grid, for instance, can be worked over again several times. TACKTIX generates "entailment rules" which are rules based solely on the relations in the grid. What will happen if these rules are integrated into the Integrated KB, and then TACKTIX is reentered with the same grid, changing it? Unless the rules have support from other sources than the grid, some or all of them will not be valid when the grid is changed[14]. This is a crucial issue for the maintenance of the Integrated KB and knowledge within KEW during the KA-process, but it is also an integration issue. Under what circumstances should that kind of knowledge be integrated?

# 7 Concluding Remarks

In this paper we have focused on integration of knowledge. We have found that integration of knowledge can be viewed as a modelling activity in the knowledge acquisition process; as iteratively refining the Integrated knowledge base by integrating pieces of knowledge acquired by the various knowledge acquisition tools in the KEW.

The integration process is an interactive process, where the KEW gives extensive support to the knowledge engineer in **ordering** of the training set, **recognition** of the relevant knowledge in the Integrated KB, **elaboration** of the consequences of introducing the new knowledge - like checking of constraints, consistency, correctness and completeness - and finally in the **adaption** of the new knowledge. These four steps constitute our computational model of knowledge integration.

We have decomposed the integration process into several subtasks and identified the main functions that are required to support each subtask. Our pilot-study showed that different representational constructs, e.g. rules and frame structures, require different operations for all the four steps in the computational model. Thus the decomposition into subtasks reflects this finding by defining one group of subtasks concerning structures and one group concerning rules.

The computational model and the identified groups of subtasks and functions provide a platform for more detailed studies of knowledge integration within ACKnowledge. Prototyping of a few integration functions has been carried out within KEW-0, giving us valuable experience. During the first half of 1990 the integration functions will be specified in detail, and implementation will start in September 1990.

[14]Let's assume that the grid is changed into a better description of the elements in the application domain.

## Acknowledgment

The work published in this paper is a part of the work done in the ACKnowledge project, and thus is based on work done by other partners. In particular we like to thank Sandra Whitehouse (GEC-Marconi Research Center) and Agnar Aamodt (SINTEF) for taking part in the formulation of a model for integration, to Geir Aakvik (SINTEF) for taking part in the reported work and to my colleagues Ingeborg Sølvberg, Inge Nordbø and Ole Jakob Mengshoel who provided valuable comments on this paper.

# References

[1] D.Needham and S.Whitehouse. The tacktix repertory grid system, version 1.0. Technical report, GEC-Marconi Research Center, 1989.

[2] B. Wielinga et. al. Conceptualization of a knowledge engineering workbench. Technical Report ACK-UvA-T1.4-DL-010-A, University of Amsterdam, 1990.

[3] G. Aakvik et. al. METAKREK : Metatool documentation. Report STF14 A88055, RIK-88-12, Sintef/Runit, Trondheim, Norway, December 1988.

[4] I. Nordbo et. al. Knowledge transformation. Technical Report ACK-STF-T2.1-DL-001-A, Sintef, Elab-Runit, Trondheim, Norway, 1988.

[5] INSTIL partners. *INSTIL Project Summary*, January 1989.

[6] A.M. Lundteigen and M. Mehus. Integration of knowledge from different knowledge acquisition tools. Report STF40 A89225, SINTEF, ELAB-RUNIT, Trondheim, Norway, December 1989.

[7] J.C. Marty, N. Shadbolt, and B. Wielinga. The acknowledge project: Improving the knowledge acquistion process. Technical Report ACK-CSI-WID-RR-001-B, Cap Sesa Innovation, 1989.

[8] K. S. Murray and B. W. Porter. Developing a tool for knowledge integration: Initial results. In *The third Knowledge Aquisition for Knowledge-based Systems Workshop*, Banff, Canada, November 1988.

[9] I. Solvberg. METAKREK : Methodology and tool-kit for knowledge aquisition. Report STF14 A88046, RIK-88-9, Sintef/Runit, Trondheim, Norway, December 1988.

# DEVELOPMENT OF SECOND GENERATION KNOWLEDGE ACQUISITION SYSTEMS

Brian Gaines

Knowledge Science Institute
University of Calgary
2500 University Drive, NW
Calgary, Canada T2N 1N4

Marc Linster

Expert System Research Group
GMD
PO Box 1240
D-5205 St. Augustin 1, FRG

Design and implementation studies are presented of the development of second generation knowledge acquisition systems through the heterogeneous integration of existing tools using inter-application protocols. The overall aim of the studies is to explore the possibility of creating highly integrated *knowledge support systems* through the loose coupling of software tools developed independently at unrelated sites for different purposes. Three tools were integrated to form a combined knowledge acquisition and performance system. The hypermedia tool *HyperCard* was used as a general-purpose knowledge acquisition tool for unstructured material in the form of text and diagrams. The knowledge acquisition tool *KSSO* was used to elicit knowledge more formally and structure it as a computable knowledge base. The expert system shell *BABYLON* was used to represent the elicited knowledge and perform inferences with it to generate advice in a performance situation. Strong synergy was created between the tools in that the annotation and explanation captured in the hypermedium system were available as context-sensitive help to the user of the expert system, and the expert system was used to validate the knowledge base generated by the knowledge acquisition tool and feed back anomalous cases as additional data for induction. An important feature of the overall system is that it is complete in its own right yet retains programmability allowing end user communities to customize it for particular requirements and domains.

## INTRODUCTION

Second generation knowledge acquisition systems targeted on applications in the 1990s have to [7]:

- 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

It seems unlikely that monolithic approaches to the development of second generation knowledge acquisition systems will be successful or that the results will be sustainable. Knowledge based system technology is developing very rapidly and involves a very wide range of computational and psychological approaches [2, 10]. This paper reports preliminary studies of building systems through the heterogeneous integration of different tools. The overall aim of the studies is to explore the possibility of creating highly integrated *knowledge support systems* through the loose coupling of software tools developed independently at unrelated sites for different purposes. The criterion for high integration is that the systems should operate as a single knowledge-based system environment from a user perspective. The underlying rationale of this research is that large-scale system development can be expedited through the integration of heterogeneous systems, each representing the best available technology in a particular application area [11].

## SYSTEM ARCHITECTURE

"Knowledge" in the artificial intelligence literature has become associated with formal structures from which inferences can be made, and knowledge acquisition and expert system tools reflect this. However, much human knowledge is not well-structured for formal representation and computer processing [6]. In the 1980s computer-based systems have been developed that are targeted on the representation of very loosely structured knowledge with less emphasis on formal structures from which inferences can be made. *Hypertext* and *hypermedia* systems approach knowledge from a different direction than that of expert systems, accepting it as having limited structure and processibility, and adding a limited structure through links that are intended to aid the user rather than allow computer-based inferencing [5, 13]. They extend knowledge representation to include any media whereby human knowledge can be transferred even if there is no means for analyzing that knowledge or its transfer.

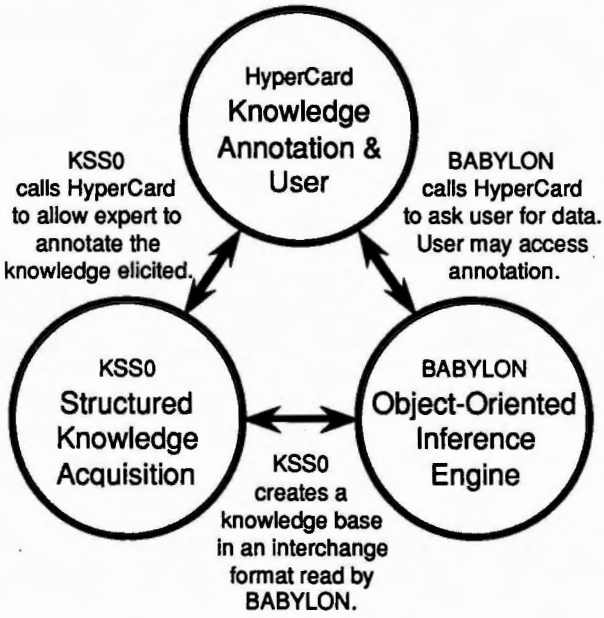



**Fig.1 Prototype knowledge support system design**

Our overall system design called for a hypermedium system to play a central role in providing for both the acquisition of informal knowledge and a flexible user interface. Figure 1 shows our initial system concept with three components:

- The hypermedia tool *HyperCard* to be used as a general-purpose knowledge acquisition tool for unstructured material in the form of text and diagrams.
- The knowledge acquisition tool, *KSS0*, to be used to elicit knowledge more formally and structure it as a computable knowledge base.
- The expert system shell, *BABYLON*, to be used to represent the elicited knowledge and perform inferences with it to generate advice in a performance situation.

It was important that the three components operate together to present the user with the appearance of a single system, and that they mutually support one another's operation. For example, the hypermedium system should provide annotation of the formal knowledge structures and the inference engine should provide validation of elicited knowledge.

The systems being integrated each have complex functionality coupled through heterogeneous integration in their own right. Figure 2 shows the overall architecture of BABYLON [4, 19] which represents knowledge in six ways, as: lisp code; free text; objects or frames; production rules; Horn clauses with the full power of prolog; and as a system of constraints [14, 15].

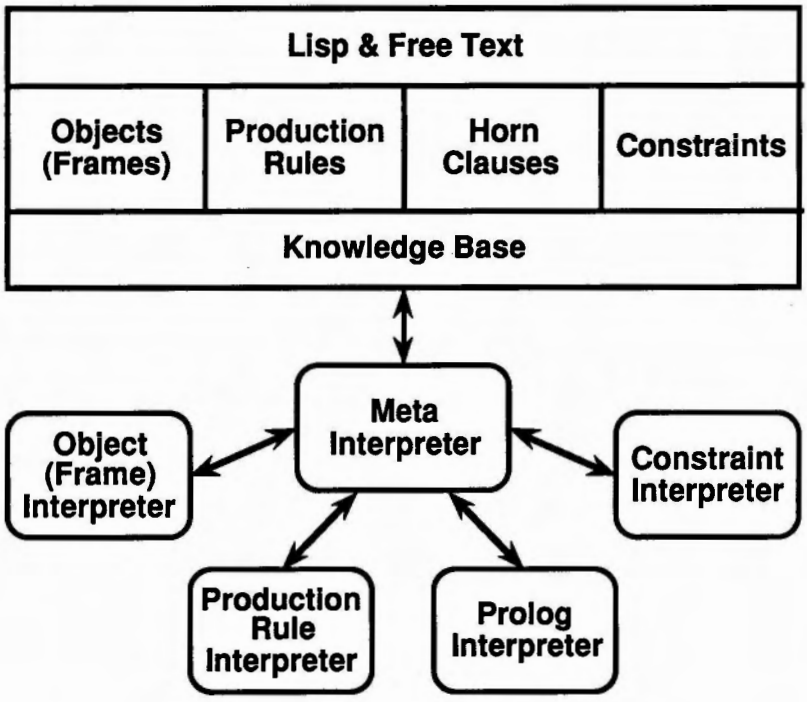



**Fig.2 The overall architecture of BABYLON**

The different representations are coupled through a meta interpreter providing a blackboard architecture through which the object, rule, prolog and constraint interpreters interact. Programmers see a uniformly available system within which they may freely mix paradigms. Table 1 shows the primary interactions possible between knowledge represented in the different forms available within BABYLON.

| Access from to | Lisp | Objects | Rules | Prolog | Constraints |
|---|---|---|---|---|---|
| **Lisp** | Direct | As slot value, in behaviors | As condition, as action | Lisp-expression as premise | Lisp-expression as value specification, activation condition, in patterns |
| **Objects** | By using messages to objects, functions for objects | Direct | Slot and behavior references serving as condition and action | Slot and behavior reference, meta predicate | Object reference as variable, value specification, activation condition, in patterns |
| **Rules** | Evaluation of rule sets | Not possible | Direct | Meta predicate | Not possible |
| **Prolog** | Proving of hypotheses | Not possible | In conditions and actions | Direct | Prolog expression as activation condition |
| **Constraints** | Definition and satisfaction of constraints | By using messages | Satisfied p-expression as premise | Satisfied p-expression as premise or in candidate | Direct |
| **Free Text** | Not possible | Not possible | As condition and action | Meta predicate | As activation condition and in patterns |

**Table 1 Interactions between knowledge in different paradigms in BABYLON**

KSSO [8] is an integration knowledge acquisition system based on an extendible set of techniques encompassing elicitation tools, visualization tools, group comparison tools, inductive tools, and export tools as shown in Figure 3.

The elicitation tools have their foundations in the personal construct psychology of Kelly [17], subsequent work on the computer elicitation of conceptual structures through repertory grids [20], and applications of these techniques to knowledge elicitation [1, 18, 21]. The objective is to gather domain knowledge from prototypical entities represented along certain dimensions:

- *Elicit* accepts specifications of entities within a sub-domain and provides an interactive graphical elicitation environment within which the experts can distinguish entities to derive their attributes. The resultant class is continuously analyzed to provide feedback prompting the expert to enter further entities and attributes.

The visualization tools consist of an interactive interface to represent the abstractions derived from those entities in terms of hierarchical clusters using Shaw's FOCUS algorithm [20], and relational diagrams such as a non-hierarchical conceptual maps derived through principal components analysis [22]. The objectives are to validate the raw domain knowledge and suggest further structure at a higher level:

- *FOCUS* hierarchically clusters entities and attributes within a sub-domain prompting the experts to add higher-level entities structuring the domain.
- *PrinCom* spatially clusters entities and attributes within a sub-domain prompting the experts to add higher-level entities structuring the domain.

The group comparison tools consist of an interactive interface to represent the relations between the terminologies and conceptual systems of different experts, or experts and clients. The objectives are to determine the consensus, conflict, correspondence and contrast between different conceptual systems [12]:

- *Socio* compares the structures for the same sub-domain generated by different experts, or the same expert at different times or from varying perspectives.

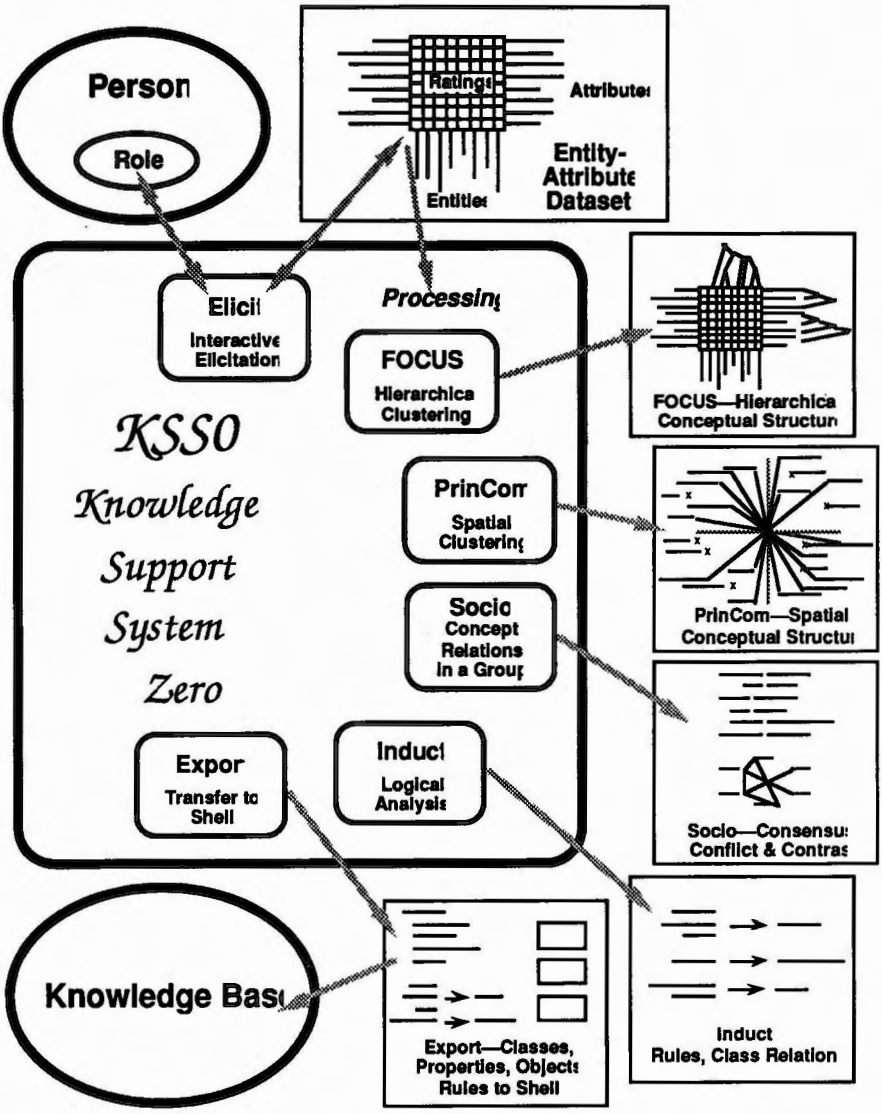



**Fig.3 The overall architecture of KSS0**

The inductive tools support the derivation of constraints within the conceptual structures through empirical induction analyses [3, 9]:

- *Induct* induces logical entailments enabling the attributes of an entity, or the evaluations of a decision-making situation in a domain, to be derived from other attributes.

The export tools support the transformation of the knowledge elicited and analyzed by tools into a formalism understandable by the shell representation and inference mechanisms:

- *Export* formats the specifications of sub-domains as classes, of entities as objects, of attributes as properties, and of entailments as methods, and transfers them to the performance tool.

## KNOWLEDGE BASE EXCHANGE

Detailed examples of the individual operation of KSS0, BABYLON and HyperCard will not be given in this paper as the focus is on the integration rather than the individual programs. One requirement for integration is that KSS0 and BABYLON be able to share a knowledge base. The basic dataset developed in KSS0 to represent an expert's conceptual structure and inference procedures consists of four parts:

a) Domain Definition: the domain, the context, the interview, the term for entities in the domain, and the term for attributes of these entities, plus the original source of these definitions if they are not original to the current dataset.

b) Attribute Definitions: the name, type and possible values of attributes, plus information about their use as inputs and outputs, and the order in which they should be requested in consultations.

c) Entity Definitions: the name and values on the defined attributes of the entities.

d) Implication Definitions: The name, test conditions and inferences of the implications.

The domain and attribute definitions are used to define a frame with the attributes as slots, together with a general instance of it in BABYLON. The entity definitions are used to define specific instances of the frame. The implication definitions are used to define rule packages in BABYLON, one for each slot whose value might be inferred.

Figure 4 shows a KSS0 dataset for Cendrowska's contact lens problem [3] in graphical form and Figure 5 shows implications derived from this dataset by Induct.

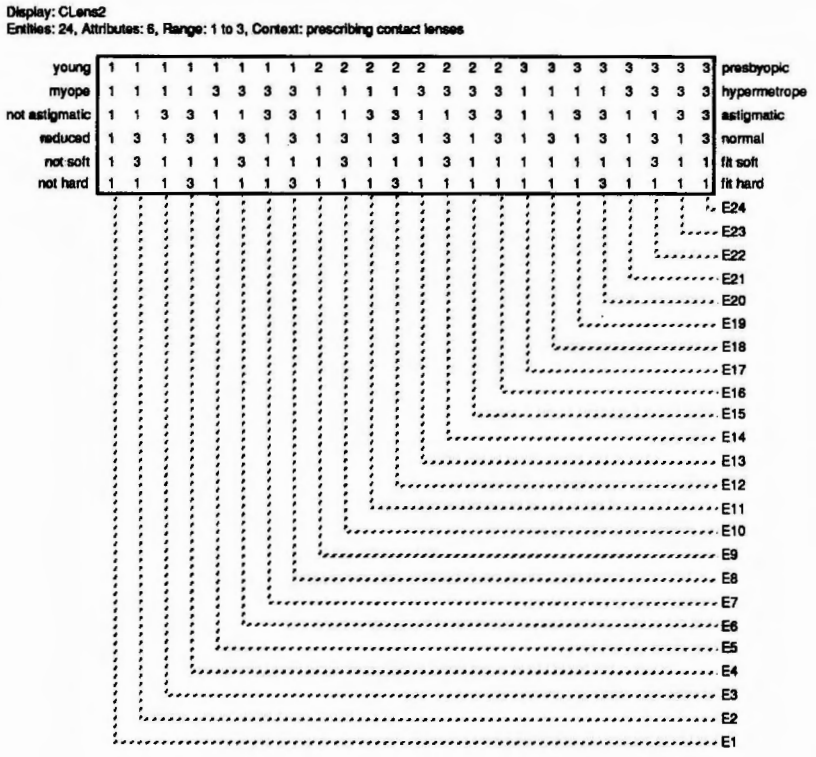



**Fig.4 KSS0 dataset**

```
astigmatism=astigmatic -> soft contact lenses=not soft 100% 6.06%
tear production=reduced -> soft contact lenses=not soft 100% 6.06%
age=presbyopic & prescription=myope -> soft contact lenses=not soft 100% 39.3%

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

astigmatism=not astigmatic -> hard contact lenses=not hard 100% 11.2%
tear production=reduced -> hard contact lenses=not hard 100% 11.2%
age=pre-presbyopic & prescription=hypermetrope -> hard contact lenses=not hard 100% 48.2%
age=presbyopic & prescription=hypermetrope -> hard contact lenses=not hard 100% 48.2%

prescription=myope & astigmatism=astigmatic & tear production=normal -> hard contact
    lenses=fit hard 100% 0.463%
age=young & astigmatism=astigmatic & tear production=normal -> hard contact lenses=fit hard
    100% 2.78%
```

**Fig.5 Implications derived from KSS0 dataset by Induct**

Figure 6 shows part of this dataset and its implications exported from KSS0. Figure 7 shows part of the dataset imported into BABYLON. Transforming the KSS0 knowledge base to the BABYLON knowledge base is automatic and involves no post-editing.

```
(KB CLens2
 (DOMAIN contact_lenses "prescribing contact lenses")

 (ATTS
  (age (CAT young pre_presbyopic presbyopic) (IO I) (WGTS 99))
  (prescription (CAT myope hypermetrope) (IO I) (WGTS 99))
  (astigmatism (CAT not_astigmatic astigmatic) (IO I) (WGTS 99))
  (tear_production (CAT reduced normal) (IO I) (WGTS 98))
  (soft_contact_lenses (CAT not_soft fit_soft) (IO O) (WGTS 99))
  (hard_contact_lenses (CAT not_hard fit_hard) (IO O) (WGTS 99)))

 (ENTS
  (E1 (ATTS young myope not_astigmatic reduced not_soft not_hard))
  (E2 (ATTS young myope not_astigmatic normal fit_soft not_hard)))

 (IMPS
  (Ind_1 (PROB 100) (ATTS ? ? astigmatic ? (not_soft) ?))
  (Ind_2 (PROB 100) (ATTS ? ? ? reduced (not_soft) ?))))
```

**Fig.6 Partial export format for dataset from KSS0 to BABYLON**

```
( Defframe People
  ( Slots
    ( Hard_Contact_Lenses - :Possible-Values
      ( :One-Of Not_Hard Fit_Hard) :Io O :Wgts 99)
    ( Soft_Contact_Lenses - :Possible-Values
      ( :One-Of Not_Soft Fit_Soft) :Io O :Wgts 99)
    ( Tear_Production - :Possible-Values
      ( :One-Of Reduced Normal) :Io I :Wgts 98)
    ( Astigmatism - :Possible-Values
      ( :One-Of Not_Astigmatic Astigmatic) :Io I :Wgts 99)
    ( Prescription - :Possible-Values
      ( :One-Of Myope Hypermetrope) :Io I :Wgts 99)
    ( Age - :Possible-Values
      ( :One-Of Young Pre_Presbyopic Presbyopic) :Io I :Wgts 99)))

( Definstance Person Of People)

( Defrule-Set :Conclude_Hard_Contact_Lenses
  ( Ind_7
    ( ?And
      ( Person Astigmatism = Not_Astigmatic))
    ( $Execute
      ( Person Hard_Contact_Lenses = Not_Hard)))

  ( Ind_6
    ( ?And
      ( Person Tear_Production = Normal)
      ( Person Astigmatism = Not_Astigmatic)
      ( Person Age = Pre_Presbyopic))
    ( $Execute
      ( Person Soft_Contact_Lenses = Fit_Soft))))
```

**Fig.7 Partial representation of dataset in BABYLON**

The BABYLON knowledge bases as shown in Figure 7 implement a heuristic association problem-solving model [16], i.e. the values of output attributes are heuristically defined on the basis of input attributes using the implication rules defined by *Induct*.

For each output attribute a rule-set is defined. A control structure fires the rule sets if their corresponding output attributes are still undefined. The firing sequence for the rules is defined by the weights of the attributes.

The operational knowledge bases are generated automatically and can be used as an expert advisory system having exactly the behavior required by Cendrowska in her original description of the dataset. In particular, BABYLON does not request information about the time-consuming measurement of "tear production" unnecessarily.

## INTER-APPLICATION COMMUNICATION

The Macintosh currently has no inter-program communication protocol and it was necessary to write code that would enable KSS0, BABYLON and HyperCard to communicate. The requirements are:

a) that KSS0 and BABYLON be able to issue a function call sending a structured message to HyperCard, and making HyperCard the active program running normally for as long as the user requires.

b) that HyperCard eventually returns to its caller, sending it a structured message in reply, and reactivating the calling program continuing as if it had returned from a function call.

c) that the three programs are able to interact asynchronously without loss of control.

An inter-application protocol was written taking commands of the form:

```
Procedure ipccall(Var toProg,fromProg,indat,outdat: Chars);
```

which accepts strings as arguments and can be called by a foreign function call from Pascal, Lisp and HyperCard. This procedure activates the program "toProg", sending it the data "indat" and asking it to return to the program "fromProg" with a reply in "outdat".

The structure of the data in "indat" and "outdat" is specific to the use of the inter-program communication. For example, to use HyperCard as a user interface for data queries from BABYLON the following structure for "indat" was defined:

Query-1,<knowledge-base>,<frame>,<instance>,<slot>,<desired-value>,<possible-values>

This information is used in HyperCard to construct a data request card as shown in Figure 8. The value on which the user clicks is returned to BABYLON in "outdat". When the consultation is complete, BABYLON may send the inferences made to HyperCard for display, or may show them within BABYLON in the usual fashion as shown in Figure 9.

The interface conforms to the normal non-modal operation of the Macintosh, and the client may browse within HyperCard, or switch to other programs using MultiFinder. In this way it is possible for the client to bring the BABYLON screen to the front without there being any return message from HyperCard.

BABYLON operates modally in a consultation and hence its screens will be inactive under these conditions. The ipccall protocol checks for any key depression and returns control to BABYLON with a null message returned. The BABYLON interface detects this message and defaults to the normal BABYLON consultation dialog.

Alternatively the user may bring the HyperCard screen to the front again and continue the dialog through HyperCard. Thus, the interface allows the maximum user freedom and never locks him or her out.

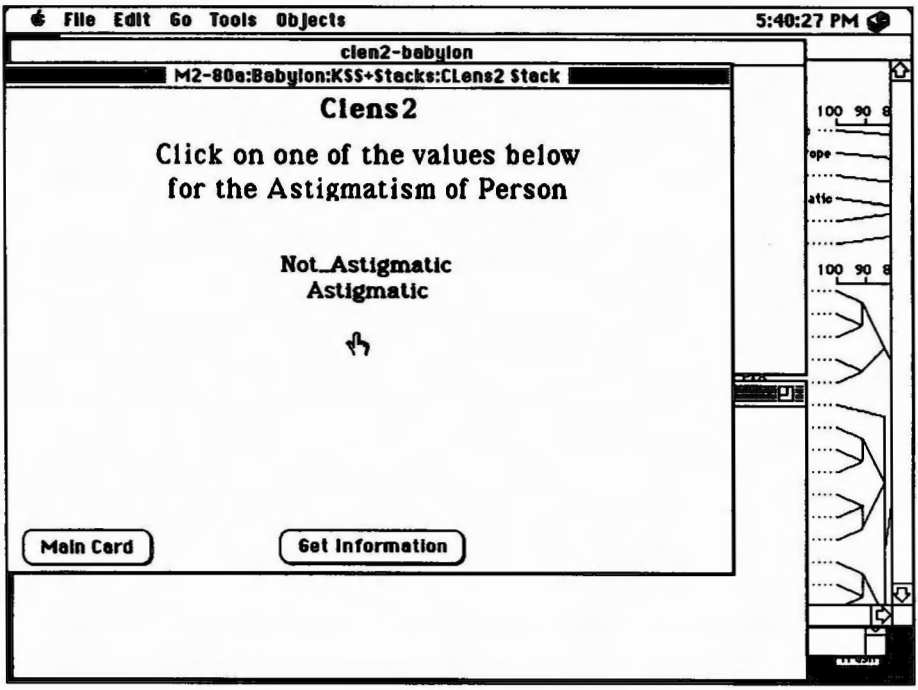



**Fig.8 HyperCard used from BABYLON to request data from client**

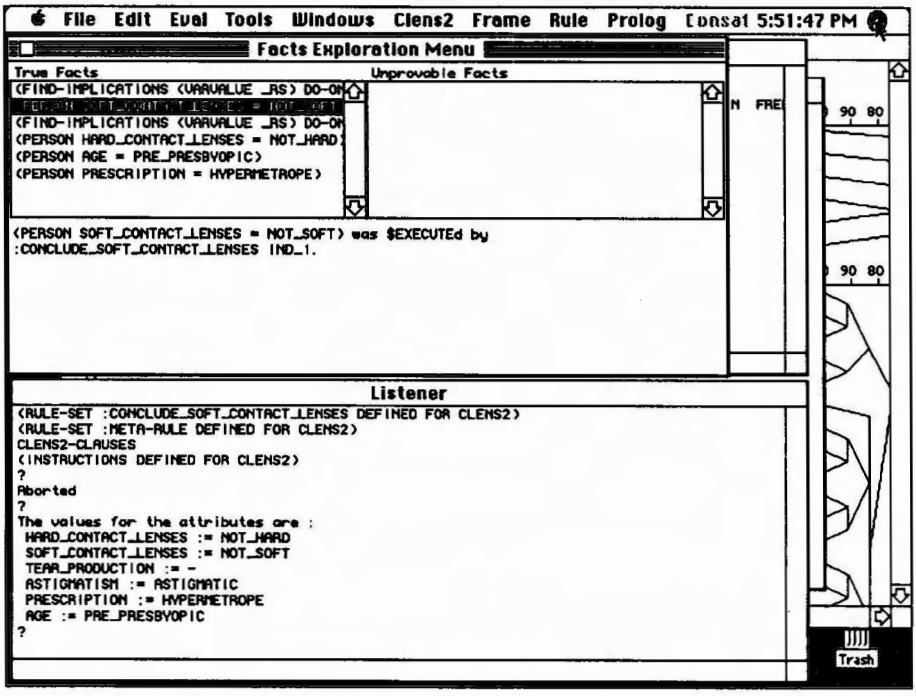



**Fig.9 Results of consultation being traced in BABYLON**

## PROTOTYPE KNOWLEDGE SUPPORT SYSTEM

KSS0 already supports interaction with HyperCard such that during knowledge acquisition an expert can transfer to HyperCard and annotate any of the entities or attribute values entered with textual notes and graphics. The stack supporting this feature automatically generates new annotation cards for each new entity and attribute value. The stack can also be used as the primary controller of KSS0, starting KSS0 if it not already running, loading datasets and sending KSS0 commands to analyze them.

The availability of the three links, HyperCard-KSS0 communication, HyperCard-BABYLON communication, and KSS0-BABYLON knowledge exchange, made it feasible to construct a prototype knowledge support system having the structure shown in Figure 1. A more detailed system design corresponding to our implementation is shown in Figure 10.

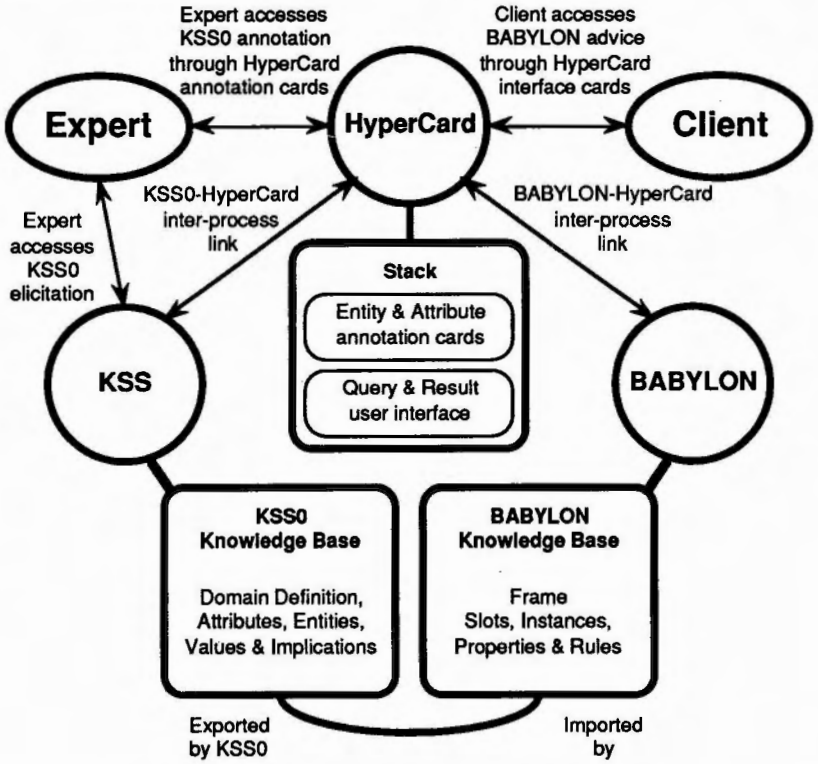



**Fig.10 Knowledge support system developed**

A generic HyperCard stack was developed with three basic cards:

1 An initial control card as shown in Figure 11 holding the dataset name and giving access to KSS0.

2 An annotation card as shown in Figure 12 used to hold notes on entities and attribute values elicited from the expert.

3 A BABYLON data request card as shown in Figure 8 used as a query interface to BABYLON during a consultation.

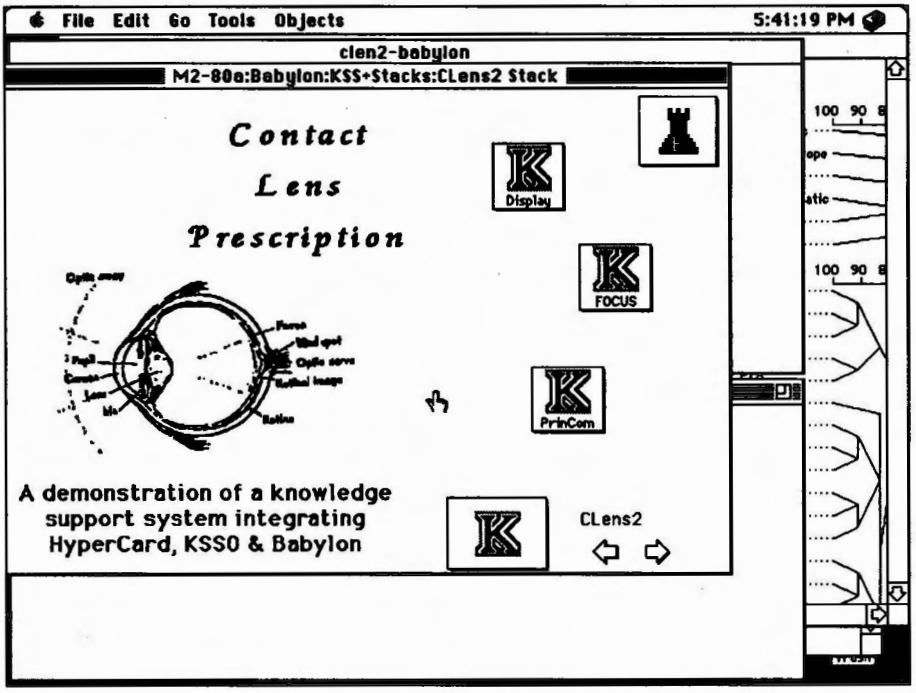



Fig.11 Initial control card

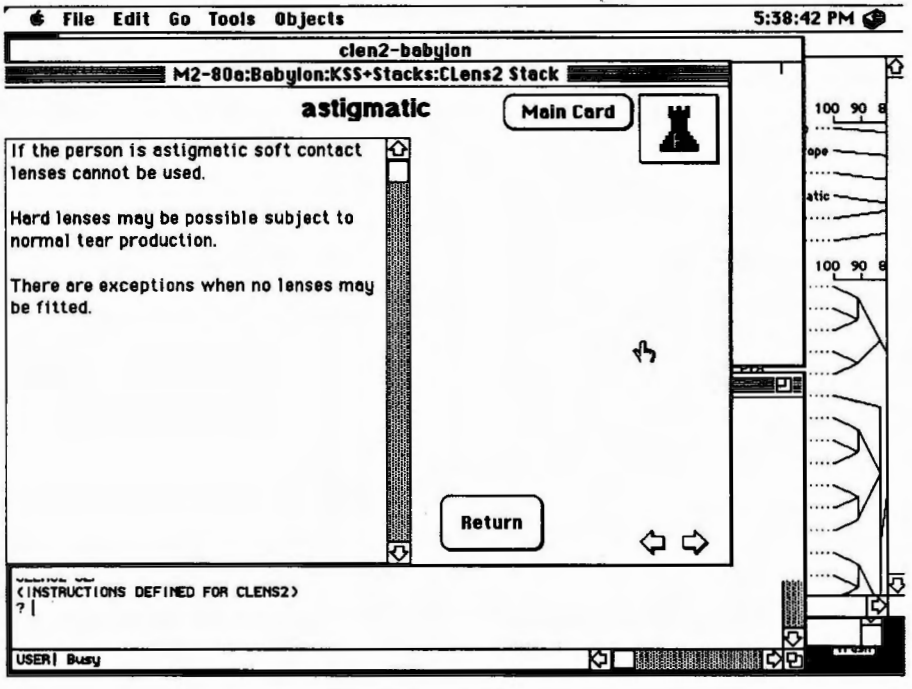



Fig.12 Annotation card

A HyperCard script was written that supports communication with both KSS0 and BABYLON. In particular it supports the "Main Card" and "Get Information" buttons shown in Figure 8. When "Main Card" is clicked control is transferred to the card shown in Figure 11. When "Get Information" is clicked the card shown in Figure 8 changes to the information request card shown in Figure 13. Clicking on one of the values now transfers control to the associated annotation card such as that shown in Figure 12. Clicking on the "Return" button returns to the BABYLON data request card (if this card had been accessed from KSS0 the "Return" button would return control to KSS0.

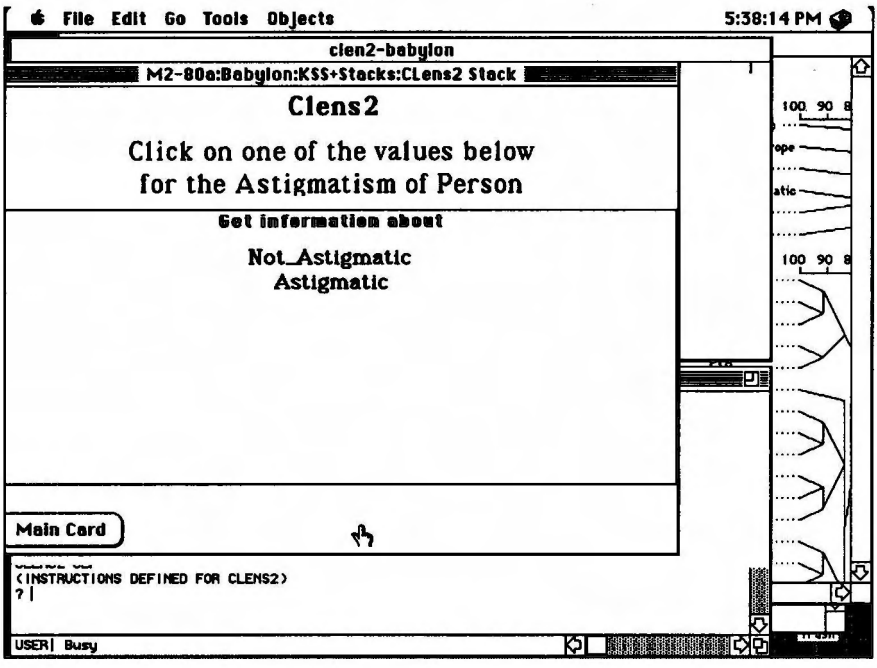



**Fig.13 Information request card**

Communication with KSS0 operates in a similar fashion. Clicking on one of the KSS0 buttons on the initial control card of Figure 11 starts KSS0 if it is not already running, loads the specified dataset into KSS0, and runs the requested analysis program if specified. Figure 14 shows a KSS0 entity match screen operating in conjunction with HyperCard and BABYLON. During knowledge acquisition with KSS0, if HyperCard is running, KSS0 makes transfer to HyperCard an option in its pop-up menus as shown in Figure 15. The availability of a pop-up menu is shown by the cursor changing to a menu symbol as it is moved over entity names and attribute value names on the screen. Depressing the mouse then brings up the pop-up menu allowing one of a number of actions to be taken, including transfer to HyperCard.

If the mouse were released over the attribute value "astigmatic" and the menu option "HyperCard" as shown in Figure 15 then control would be transferred to the HyperCard annotation card shown in Figure 12. The expert can then annotate the intended use of the term "astigmatic" with text and diagrams, and return control to the KSS0 screen of Figure 15 by clicking "Return". The expert can also browse the HyperCard stack at will, reading and modifying other annotation, before returning control to KSS0. All KSS0 screens support the pop-up menu communication with HyperCard giving access to the annotation from, for example, the PrinCom conceptual cluster screen as shown in Figure 16.

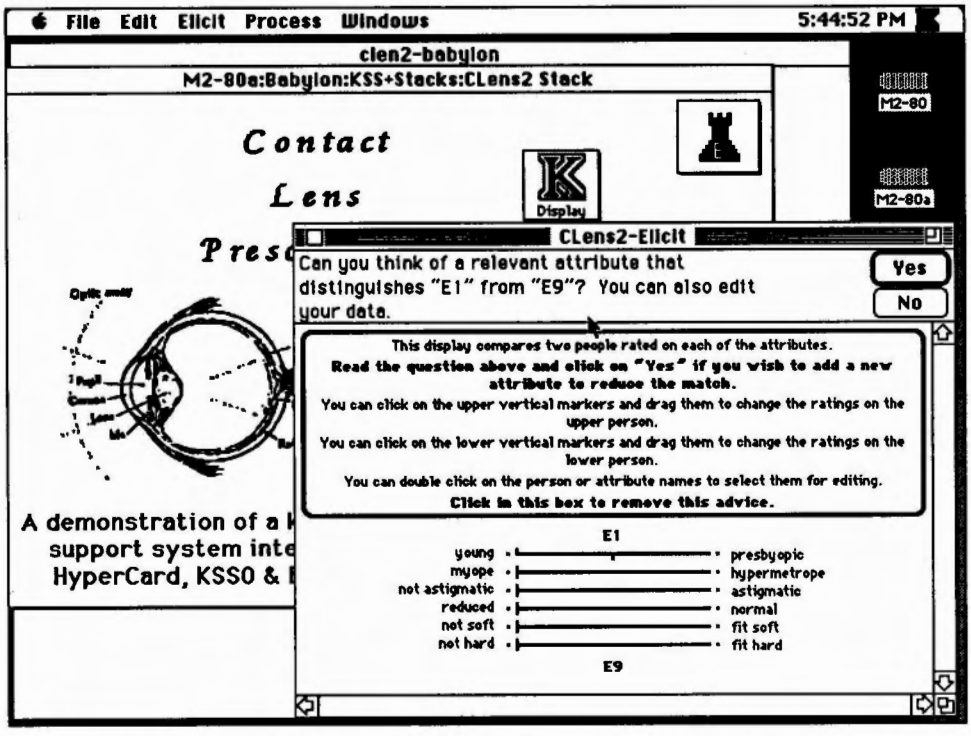



**Fig.14 KSS0 operating in conjunction with HyperCard and BABYLON**

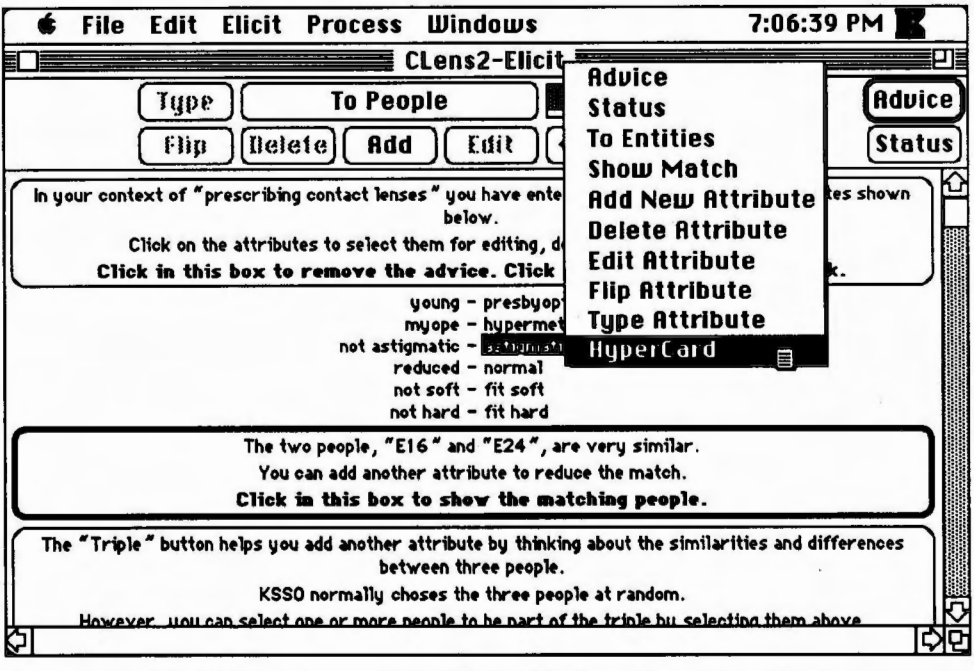



**Fig.15 Pop-up menu in KSS0 allowing transfer to HyperCard**

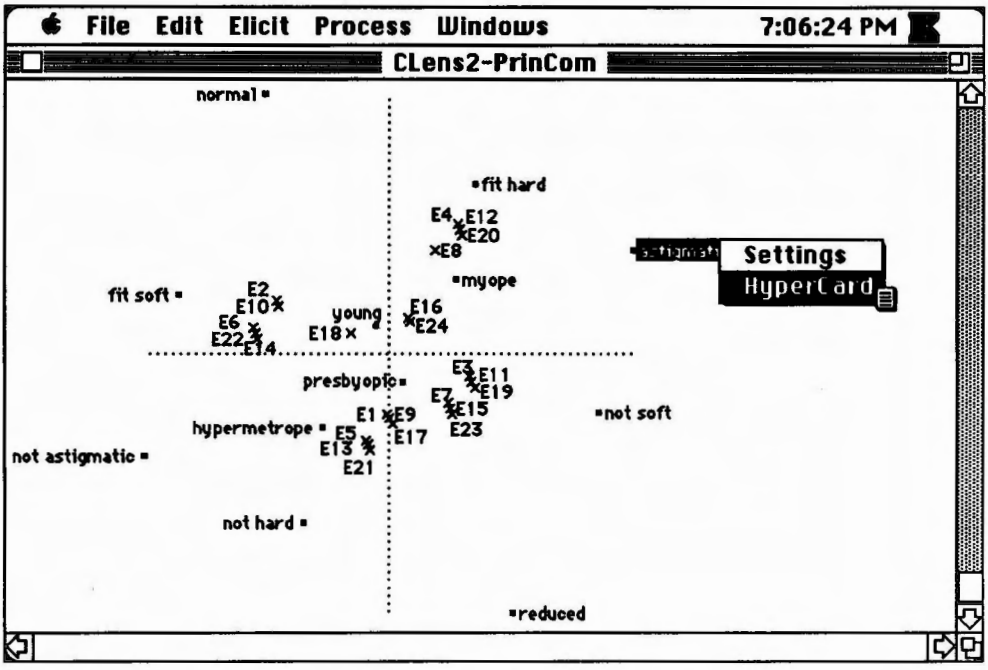



**Fig.16 Data clustering in KSS0 with access to HyperCard**

The close coupling between KSS0 and BABYLON enables the expert system shell to be used as a performance tool validating the knowledge base created in KSS0 against test cases entered through BABYLON. If the decisions recommended for the test cases are valid then no action need be taken, but if they are not the expert can transmit them from BABYLON to KSS0 as new data. This will affect the results of conceptual clustering and induction, and hence the knowledge base that is exported to BABYLON. Dependent on the amount of data and the difficulty of the induction, either the new knowledge base will operate correctly or further data may be posted back to KSS0 until it does so. Thus the the coupling of the acquisition tool and the expert system shell results in new modes of acquisition.

## CONCLUSIONS

The studies and developments reported show the feasibility of creating highly integrated knowledge support systems through the loose coupling of software tools developed independently at unrelated sites for different purposes. This is an important direction for future research and development of knowledge-based systems. The following analysis examines the roles played by each component and suggests directions for further development.

The hypermedium system plays two major roles in the system. First, it acts as an annotation facility allowing background knowledge and explanations to be captured from the expert that could not otherwise be entered in the structured knowledge acquisition tool. Second, it acts as a programmable user interface to the expert system shell. These two roles are synergistic in that the background knowledge and explanations entered during the acquisition phase are available to the user, and he or she may also extend them with further notes as a result of application experience.

The availability of the programmable hypermedium system has practical significance in giving end user groups the capability to customize the system for particular requirements and domains. It also has theoretical significance in allowing the entry of knowledge that does not fit the computational framework of the acquisition system or shell. This increases the possibility that the resultant system will be complete in that it enables information that goes beyond the formal knowledge base to be entered. Completeness cannot be checked within the formal system. It requires analysis of the knowledge base in the light of the doubts and reservations expressed by the expert and users without the constraints of computationally usable knowledge.

The expert system shell plays two major roles in the system. First, it performs its normal role of providing an advisory system based on the knowledge base. Second, it provides an on-line validation capability for the knowledge acquisition system. The knowledge acquisition system plays two major roles in the system. First, it performs its normal role of rapid prototyping in developing the knowledge base. Second, it provides a continuing on-line acquisition capability for the expert system.

The main limitation of the current system is that edits of the knowledge base in the expert system shell cannot necessarily be propagated back to the knowledge acquisition tool. That is, it is possible to create data and control structures that do not correspond to those created by the acquisition tool. Thus, the continuing applicability of the acquisition system during the application of the knowledge base is dependent on its not being edited within the shell. This is similar to the usual problem of editing the machine code generated by high level languages compilers. However, accepting the loss of editing within the shell would place very strong requirements on the knowledge acquisition system to be capable of representing all aspects of the domain and problem-solving knowledge.

The effort involved in creating the system reported in this paper was two person-weeks, part of which went into the inter-application protocol, part into the HyperCard scripts and part into the knowledge base export and import. The comparable effort to create a monolithic system would be many person-years. The heterogeneous integration strategy is an effective way to develop a next generation of knowledge-based systems.

As inter-application protocols become better standardized and more uniformly available, for example in X-Windows and Apple's System 7, it will become advantageous to split systems into modules at an even finer level of granularity than we have described here. BABYLON already consists of a number of distinct interpreters communicating through a blackboard that could well become a light-weight inter-application protocol. Similarly, KSS0 consists of a number of distinct knowledge acquisition tools operating around a common knowledge base.

Heterogeneous integration enables the full power of tools to be placed in the hands of end user groups without their losing the essential programmability of computer systems. The frame does not become a cage! We see future knowledge based systems as a loose amalgam of programmable modules presenting to the end users the feel and appearance of a highly integrated system, yet preserving for the end user communities the programming capabilities to customize the system as much as they require.

## ACKNOWLEDGEMENTS

We are grateful to our colleagues at the GMD, KSI and Knowledge Acquisition Workshops for many stimulating discussions which have affected the research reported in this paper. Research in the Knowledge Science Institute is funded in part by the Natural Sciences and Engineering Research Council of Canada.

## REFERENCES

[1] Boose, J.H. & Bradshaw, J.M. (1987) Expertise transfer and complex problems: using AQUINAS as a knowledge acquisition workbench for knowledge-based systems. **International Journal of Man-Machine Studies 26** (1), 3-28 (January).

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

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

[4] Christaller, T., di Primio, F. and Voß, A. (1989) **Die KI-Werkbank BABYLON**. Bonn: Addison Wesley.

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

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

[7] 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).

[8] Gaines, B.R. (1988b). Knowledge acquisition systems for rapid prototyping of expert systems. **INFOR, 26**(4), 256-285 (November).

[9] Gaines, B.R. (1989a). An ounce of knowledge is worth a ton of data: quantitative studies of the trade-off between expertise and data based on statistically well-founded empirical induction. **Proceedings of the Sixth International Workshop on Machine Learning.** pp.156-159. San Mateo, California: Morgan Kaufmann (June).

[10] Gaines, B.R. & Boose, J.H., Eds. (1988). **Knowledge Acquisition for Knowledge-Based Systems**. London, Academic Press.

[11] Gaines, B.R., Rappaport, A.T. & Shaw, M.L.G. (1989) A heterogeneous knowledge support system. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Fourth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop**. pp.13-1-13-20. Banff (October).

[12] Gaines, B.R. & Shaw, M.L.G. (1989). Comparing the conceptual systems of experts. **Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.** pp.633-638. Los Angeles, California: Morgan Kaufmann (August).

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

[14] Guesgen, H.-W. (1989). **CONSAT: A System for Constraint Satisfaction**, London: Pitman Publishing.

[15] Guesgen, H.-W., Junker, U. & Voß, A. (1987) Constraints in a Hybrid Knowledge Representation System, **Proceedings of the Tenth International Joint Conference on Artificial Intelligence**, pp 30 - 33. Los Altos, Ca.: Morgan Kaufmann.

[16] Karbach, W., Linster, M. & Voß, A. (1990). Model-based Approaches: One Label - One Idea?, introduced to **EKAW90**, Amsterdam

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

[18] Linster, M. (1989) Towards a second generation knowledge acquisition tool. **Knowledge Acquisition, 1**(2) 163-183 (June).

[19] di Primio, F. and Wittur, K. (1987) BABYLON, a Meta-Interpretation Model for Handling Mixed Knowledge Representations. **Proceedings of the 7th International Workshop on Expert Systems and Their Applications**. pp. 821 - 833. Avignon.

[20] Shaw, M.L.G. (1980). **On Becoming a Personal Scientist: Interactive Computer Elicitation of Personal Models of the World**. London: Academic Press.

[21] 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).

[22] Slater, P., Ed. (1976). **Dimensions of Intrapersonal Space: Volume 1**. London: John Wiley.

# Validation of User Intention

Thomas Hoppe*
LIACC - Laboratório de Inteligência Artificial e Ciências de Computação
Universidade do Porto, Rua de Campo Alegre 823, 4100 Porto, Portugal
E-mail: thoppe@liacc.up.ctt.pt

**Abstract**

Knowledge validation has recently received greater attention. What seems to be evolving is an intimate relationship between knowledge acquisition and knowledge validation. Like knowledge acquisition, knowledge validation can be viewed from different viewpoints. One viewpoint is based on the assumption that the user has a clear idea of the intended system. Under this assumption knowledge validation means to determine whether the actual system meets the user intention. From another viewpoint we must take into account that in the modeling process the user's idea of the intended system is evolving. If we accept this assumption, knowledge validation means to determine whether the user intention meets the actual system.

## Introduction

Knowledge validation has recently received greater attention. Whereas in the past only little efforts were made to ensure the quality of knowledge and of knowledge based systems this situation has changed in the last years. Not only the terminology of this young field is evolving rapidly [1,9,17], but also new projects were launched [10] and new methodologies were proposed [3,7,11,19]. Not only in the field of knowledge validation, but also in the fields of knowledge elicitation [23] and knowledge acquisition [13] it was recognized that their exists an intimate relationship between knowledge acquisition and knowledge validation.

Evolutionary knowledge acquisition is usually distinguished from traditional software engineering, because it has to handle situations where experts have difficulties in expressing specifications of expertise. The work of Morik [13] has shown that knowledge acquisition can be viewed from two different viewpoints, the 'bottleneck' and the 'sloppy modeling' viewpoint. From the 'bottleneck' viewpoint the acquisition process is regarded as a transfer of already existing knowledge from the user to the system, where either the expert or the system can be regarded as the "bottle". In the context of 'sloppy modeling' Morik has recognized that not only the formalisation is evolving in the modeling process, but also the model the user has in mind. Similar viewpoints can be adopted for incremental knowledge validation.

A viewpoint similar to the 'bottleneck' view is based on the assumption that the user has a clear idea of the intended system. Under this assumption knowledge validation means to determine whether the actual system meets the user intention. The other viewpoint assumes

*This work was supported by a grant from the German Academical Exchange Service (DAAD). I like to thank Pavel Brazdil for some comments concerning inverse resolution and for the spontaneous invention of reconstructability as a termination criterion. Also I would like to thank the two unknown refeeres for their valuable comments.

that the user's idea of the intended system is evolving. This is similar to the 'sloppy modeling' view. If we accept this assumption, knowledge validation means to determine whether the user intention meets the actual system.

Of course, neither of these assumptions in isolation is right. Knowledge validation integrated into an incremental knowledge acquisition environment must take both assumptions into account. Not only the knowledge of the system can be wrong, but also the idea of the intended system in the mind of the user.

Before we are able to address the problems of both assumptions in an incremental knowledge validation environment, we will address the problem of validating the user's intention in isolation first. Although this problem seems to be difficult, some results indicate that there exists a solution.

In the following I give a short overview about the used terminology (Section 1), give a preliminary formal definition of user intention (Section 2), show a connection between knowledge validation and machine learning (Section 3), describe some common modeling errors (Section 4), discuss some problems connected with validation by experimentation (Section 5), point to a new approach to overcome these problems (Section 6) and show the result of an experiment which confirms this approach (Section 7).

# 1 Verification, Evaluation and Validation

O'Keefe et al. [17] investigated different forms of testing under the topic 'validation' and pointed to the difference between verification and validation using the following characterization:

Verification means to build a system right.
Validation means to build the right system.

Benbasat & Dhaliwal [1] introduced more precise definitions for validity and verification, and distinguished further between verification, validation and evaluation[1]. We rephrase their definitions as follows:

**Verification** is the proof of certain logical properties (e.g. consistency, correctness and completeness) in the development life cycle of a system[2].

**Validation** is the determination of a homomorphism between a system and its representation.

**Evaluation** is the measurement of certain properties (e.g. quality, utility and usability) of a system.

Benbasat & Dhaliwal [1] discriminated knowledge validation techniques according to the stages of the development life cycle of a system. They introduced a global classification framework for the different notions of knowledge validation. In their terms 'knowledge acquisition validation' subsumes 'conceptual validation' and 'elicitation validation', and covers the modeling process from the conceptualization of a domain to an elicited knowledge base. 'Implementation validation' takes place in the system construction phase and ensures the validity between the implemented and the elicited knowledge base[3]. 'Knowledge-based system validation' subsumes 'functional evaluation' of an implemented knowledge base against a nonexistent, idealized knowledge base and the 'representational validation' against the user.

[1] This distinction was also made by de Mántaras et al. [10].

[2] We use the term 'system' in a broad sense and cover with it 'formalisation', 'representation', 'knowledge base' and 'knowledge based system'.

[3] Benbasat & Dhaliwal [1] termed 'implementation validation' also 'verification'. This not only seems to be the source of some terminological inconsistencies in their work, but it also blurs the distinction between a formal proof and the determination of a homomorphism.

de Mántaras et al. [10], López et al. [9] and Hoppe [7] take a slightly different view which is not so much concerned with the distinction of different validation processes in the development life cycle. Their terminologies[4] reflect the methodologies used for the validation of a system. A unified classification framework they propose distinguishes between:

**Inspection:** Unsupported walkthroughs through system structure and behavior.

**Explanation:** Inspectable justifications of the system behavior.

**Functional Validation** which can be further subdivided into:

**Testing:** Checking the behavior of a system in a random way.

**Experimentation:** Checking the behavior of a system in a systematic way.

**Structural Validation:** Checking logical properties of a system.

In the following we will investigate the process of 'representational validation' in the background of an imperfect user in more detail and discuss some of the problems which arise if we validate the user intention by experimentation. Before we define the notion of 'user intention', we briefly recall some model-theoretical terms needed for the discussion below.

We assume a first-order *language* build out of *constant symbols*, *function symbols* and *relation symbols*. We assume further the traditional *logical connectors* and *quantifiers*. *Clauses*, *literals* and so on have their usual meaning. A set of clauses is called a *theory*.

Further we assume a set of *individuals* called *domain*. Together with a *functional base set* and a *relational base set* the domain builds a triple which is called *conceptualization*.

We need further *interpretation functions* to give our theories semantics. A *pre-interpretation* maps constant symbols into individuals, function symbols into elements of the functional base set, and relation symbols into elements of the relational base set. A *variable assignment* maps variables into individuals. An *interpretation* is a mapping from the language into the conceptualization and consists of a pre-interpretation and a variable assignment. A *model* of a theory is a set of true ground unit clauses deducible from the theory.

# 2 User Intention

If we talk about the user intention, we should of course clarify what we mean by this term. For this purpose we use the definition of interpretation functions from model theory as a starting point. We orientate our definition of user intention on the definition of intended interpretations[2,8,24].

**Definition 1** *An* intended interpretation $f_I$ *is a mapping from elements of a language into elements of a conceptualization, where* $f_I$ *is assumed to be fixed.*

This is a common definition used in logic programming. The assumption that the interpretation function is fixed is usually realized by the assumption that there exists an "oracle" which always decides the truth of clauses correctly. We will review this assumption in the next section. Because we are interested in imperfect users, we extend the definition of intended interpretation in the following way:

**Definition 2** *An* user intention $f_U$ *is a mapping from elements of a language into elements of a conceptualization, where* $f_U$ *is not necessarily assumed to be fixed.*

[4] de Mántaras et al. [10] and López et al. [9] called functional validation 'verification of the KBS functionality' and structural validation 'structural verification'. We should not blur the distinction between verification and validation, because otherwise we are confronted with the unsolvable problem of proving the consistency of a system.

On the basis of user intention we define the notions of correctness, completeness and rightness with respect to the user intention, where M(Th) is a model of the theory:

**Definition 3** *A theory Th is* correct *w.r.t.* $f_U$, *iff* $M(Th) \subseteq f_U(Th)$

**Definition 4** *A theory Th is* complete *w.r.t.* $f_U$, *iff* $M(Th) \supseteq f_U(Th)$

**Definition 5** *A theory Th is* right *w.r.t.* $f_U$, *iff* $M(Th) = f_U(Th)$

These definitions have on one hand the advantage that already existing results and systems concerning correctness and completeness (like e.g. debugging methods [22], consistency and completeness checker [3,11,19,25]) can easily be adopted for checking logical properties in an incremental knowledge validation environment. We only have to assume that the user intention does not change in a structural validation step, because a fixed intended interpretation is just a special case of a possibly changing user intention.

On the other hand, these definitions seem to be sufficient to describe and handle problems connected with changing interpretation functions in incremental knowledge validation systems.

# 3 Knowledge Validation and Machine Learning

If we look at the trends in machine learning, then we can observe that 'constructive induction' is one of the hot topics. Most of the work on constructive induction has its roots in the work of Shapiro [21,22]. Muggleton & Buntine [16] introduced 'inverse resolution' to solve one of the main problems of constructive induction, the 'new term problem'. Following this line Wirth [26,27] and Rouveirol & Puget [20] presented recently extensions and simplifications of inverse resolution. de Raeth & Bruynooghe [18] and Wrobel [28] used meta-knowledge in form of rule-schemes and rule-generating rules to tackle the same problem from the meta-level.

Most of the above approaches rely on an "oracle[5]", which judges induced rules and invented new terms or concepts. The assumption is made that the user (the oracle) always judges the invented knowledge on the basis of an intended interpretation correctly.

Of course, for limited domains (like a blocks world) or for well-known domains (like family relations), where we can expect that the user has experience with the domain, this assumption holds. But if we consider yet unknown domains (like AIDS therapy), large domains (like infectious diseases) or a domain with an ill-structured conceptualization (like the new political east-west relations) the "oracle" assumption seems not to be adequate.

For example, in an experimental inverse resolution system[6] currently under development the acquisition sequence shown in figure 1 occured in the course of acquiring 'save_to_stack' rules from examples. o1 and o2 represent objects and 'weight' the relation between an object and its weight. We assume that all objects of weight 5 are endtables. The induced Prolog clauses are implicitly universally quantified and user input is underlined.

The meaning of the rules found by the absorption operator is that the weight of object o1 (respectivly o2) is the same as the weight of all other objects with the same weight. The user found this obviously true and accepted the proposed clauses, implicitly assuming a closed-world. Subsequently the system invented a new predicate. The user named this predicate 'endtable' and gave the system one more fact about a non-endtable, now working under the open-world-assumption. The resulting knowledge state of the system is shown in figure 2.

At the first sight this knowledge state seems to be what the user intended. But surprisingly for the user, this knowledge state produces an inconsistency, if the system is asked whether *weight(o1,2)* is true. The inconsistency stems from the fact that the user did not recognize that the object denoted by variable C in the intra-construction step must also belong to the same class as the object denoted by variable A.

[5] Shapiro [21,22], Muggleton [14], Muggleton & Buntine [16] and Wirth [26,27] call a perfect user "oracle". This "mystical" idealization seems not to be well-suited for incremental knowledge acquisition in general.

[6] More details about inverse resolution can be found in the already mentioned references

```
⟹ weight(o1,5).
⟹ weight(o2,5).
ABSORPTION found rule: weight(o1,A) :- weight(B,A).
Is this rule always true ? yes.
ABSORPTION found rule: weight(o2,A) :- weight(B,A).
Is this rule always true ? yes.
INTRA-CONSTRUCTION invents predicate:
    PRED(o1) :- true.
    PRED(o2) :- true.
    weight(A,B) :- weight(C,B), PRED(A).
Are these rules always true ? yes.
How shall I call PRED ? endtable.
Rule endtable(o1) :- true added to theory.
Rule endtable(o2) :- true added to theory.
Rule weight(A,B) :- weight(C,B), endtable(A) added to theory.
⟹ weight(o3,2).
```

Figure 1: Acquisition sequence

```
endtable(o1).
endtable(o2).
weight(A,B) :- weight(C,B), endtable(A).
weight(o3,2).
```

Figure 2: Knowledge State

Of course, all the questions asked by the system are correct under the closed-world-assumption w.r.t. inverse resolution and the given information. In this case the user was wrong when he accepted the rules proposed by the system. If we now imagine that in some situations the user is asked to judge the rightness of more complicated, invented predicates (e.g. with an arity higher than 2), it becomes obvious that it is more difficult for the user to interpret the proposed knowledge. Hence the user can make errors more easily. In this case a constructive induction or knowledge acquisition system can not rely anymore on an "oracle", it must take an user into account who is sometimes wrong.

# 4 Common Modeling Errors

In the context of 'sloppy modeling' Morik [13] detected four common types of faults users make in the modeling process. On the example of jaundice she explained the following errors:

**Case of the unbound variable:** In this case a variable connecting the premisses of a rule with its conclusion is missing, e.g. disturbed(X) :- color(Y). This can be corrected by the introduction of a new premisse, like e.g. symptom(X,Y).

**Case of an missing argument:** The rule disturbed(X) :- color(Y), symptom(X,Y) refers implicitly to only one person, so all facts entered either. Here a qualification over a set of persons is needed, which can be expressed by relating the color and disturbed informations to a person: disturbed(X,Y) :- color(X,Z), symptom(Y,Z).

**Case of the interesting constant:** In the course of the modeling process the user can detect that the knowledge formalized until now as constants (i.e. liver and yellow) is not sufficient to express certain knowledge and that instead the use of predicates would

be better (e.g. liver_disturbed(X) and yellow(X)). This knowledge can be introduced by the rules: yellow(X) :- color(X,yellow) and liver_disturbed(X) :- disturbed(X,liver).

**Case of the missing predicate:** In this case a rule is overgeneral, because of a missing predicate and hence different cases cannot be discriminated. This can be corrected by adding a predicate which specializes the rule.

There are of course some other modeling errors that can be detected. Because we think that any validation tool should handle as much modeling errors as possible, we extend the above list by the following cases detected in our experimental inverse resolution system.

**Case of the argument order:** Let us look at an example. How should we interpret the fact *mother(sabine,kim)*? Is Sabine the mother of Kim? Or has Sabine as mother Kim? Usually, someone adopts a personal rule for the conceptualization of a domain (Brazdil, pers. comm.), if he abbreviates the names of predicates, like for example the usual subject-predicate-object order of an equivalent natural-language sentence.

The things get a little bit more complicated, if we formalize yet unknown domains or if a system invents new predicates. In the first case it happens sometimes that we detect in the course of the conceptualization that a chosen argument order for a predicate was not appropriate. In the second case it is clear that the system does not care about the argument order. But the user does, if he interprets the proposed knowledge to determine its meaning.

**Case of diverging assumptions:** In our example the user interpreted the knowledge state under the closed-world-assumption, but worked with the system under the open-world-assumption. A lack of knowledge about the restrictions and underlying assumptions of a system can be a further source of errors.

**Case of the unknown consequences:** Let us look again at the example of section 3. At the point where the user decided to accept the rules, he did not knew the consequences of his decision. Only after he run into the "trap", it became clear that the modeling decision was erroneous.

The case of the 'missing predicate', the case of the 'argument order' and the case of the 'unknown consequences' occur frequently in inverse resolution systems, in section 7 we will see some more examples. These cases represent "natural traps" which are useful for knowledge validation by reconstruction. But before we introduce this method, we investigate the problems of acquiring the user intention by experimentation.

# 5 Validation by Experimentation

The main problem of validating the user intention is of course the user intention. Not only because it can be wrong as the last sections indicated, but also because there exists not yet an acquisition method for the user intention. In this section we will point out the problems connected with hypotheses generation by partial evaluation [7], if we use experiments to acquire the user intention.

v. Harmelen & Bundy [5] and Ginsberg [4] reported that explanation-based generalisation [12] is a subset of partial evaluation. Explanation-based generalisation and partial evaluation are based on the distinction of *theoretical terms* and *observational terms* [7]. Here the term *theoretical term* refers to predicates, which appear only as positive literals in the clauses of a theory and the term *observational term* refers to predicates, which appear only as negative literals in the clauses of the theory. The idea behind both approaches is to

[7] This distinction was introduced by Hempel [6] into the philosophy of science and by Ginsberg [4] into the field of knowledge base refinement. In EBG terminology, observational terms are required to be operational.

```
eval(Goals,Th,GPM) ⇔
    FOR ∀ Gi ∈ Goals DO
        IF  ∃ Cl ∈ Th: unify(Gi,head(Cl))
            THEN eval(body(Cl),Th,GPMi)
            ELSE IF build-in(Gi) ∧ evaluable(Gi)
                    THEN call(Gi)
                    ELSE GPMi := {Gi}
    ENDDO
    GPM := ⋃ GPMi
           i
```

Figure 3: Partial Evaluator

compile theoretical terms of a theory into their observational equivalents. Explanation-based generalisation uses a background theory to compile an example into a more general rule which can be used for more efficient derivations of the example. Partial evaluation computes all possible rules without an example.

Hoppe [7] introduced experiments for the purpose of systematical knowledge validation based on partial evaluation. Experiment hypotheses consist of observational predicates which form partial models of a theory. In the partial evaluation process (see figure 3) the complete derivation tree of a theory is expanded as long as clauses for theoretical terms are applicable. If no clause is applicable for a predicate, an observational leaf node was found, which is delivered back. Every possible proof path corresponds to a set of observational predicates. These sets are called generalized partial models (GPM).

GPMs describe classes of models covered by the theory. They can be used to perform instantiation experiments against a set of testcases, with another system or the user. At first sight instantiation experiments seem to be useful to acquire the user intention (a mapping of variables into individuals of a conceptualization), but if we look more carefully at the hypotheses generation, we detect that we are confronted with the following problems:

1. With instantiation experiments we acquire an interpretation function extensionally. We cannot perform instantiation experiments with the user to acquire all possible instantiations, because the number of models covered by a theory is infinite.

2. The approach restricts the GPMs to the most special partial models, i.e. all predicates occuring in a GPM must be observational. This means that by instantiation experiments we can only acquire the user intention for observational terms. For theoretical terms we cannot acquire the user intention, because by partial evaluation they are compiled away.

3. A modified approach which determines mixed GPMs (which contain observational and theoretical terms) is because of its computational complexity not practicable.

   In the worst case, if the theory contains n observational predicates we have to compute $2^n$ GPMs. The power set of the set of observational terms forms under the subset relation a complete lattice. The number of sets of observational terms in every level of the lattice can be determined by the number of combinations without repetition. So the number of all mixed GPMs is:

$$O(n) = \sum_{k=0}^{n} \binom{n}{k} = 2^n$$

   where n is the number of observational terms and k is the level. In general, it is difficult to determine the number of mixed GPMs for an arbitrary theory, because dependent on the theory the number of possible combinations of observational predicates and thus the number of GPMs are reduced.

```
stratified_bmgs(Th,i,BMGs) ⇔
    IF  Th = ∅
        THEN BMGs = ∅
        ELSE  BMGs_i := {<i,x> | x ∈ Th ∧ bottom_most_goal(x)}
              Th_i := Th \ {x | x ∈ Th ∧ bottom_most_goal(x)}
              stratified_bmgs(Th_i,i+1,SBMG)
    ENDIF
    BMGs := BMGs_i ∪ SBMG
```

Figure 4: Determination of stratified BMGs

4. Recursive theories cause problems for partial evaluation, because of infinite branches in the derivation tree. This stems from the fact that the leaf nodes of the derivation tree are never instantiated, thus a depth-first strategy has no possibility to terminate on infinite branches. This can be avoided with a breadth-first strategy, which maintains a list of already visited goals, thus avoiding derivation cycles. If a derivation cycle was detected, a careful depth-restricted evaluation can proceed on that branch. This solution is practicable, but it leaves one question open: When to stop the evaluation on an infinite branch?

# 6 Validation by Reconstruction

We can avoid some of the problems determined in the last section with a modified experimental validation technique called validation by reconstruction. In the following we explain the general idea of validation by reconstruction, describe its basic algorithms and sketch a plausible algorithm for this new approach. The basic algorithms were used for an exemplifying experiment which is described in the next section.

What is interesting in a recursive theory is that it expresses potentially infinite information intensionally. Thus, if we evaluate a recursive theory on an infinite branch, we obtain GPMs which grow in size and/or share a similar predicate structure. Suppose now that we have a mechanism which "reconstructs" a theory from its GPMs, then we can stop the generation of experiment hypotheses, if we obtain the original theory from the "reconstruction". Hence our termination criterion for recursive theories can be described as: Generate GPMs as long as it is not possible to "reconstruct" the original theory, otherwise terminate.

Let us make this idea more precise. We call a literal which appears as positive literal in a clause whose negative literals did not occur as positive literal in any other clause a bottommost goal (BMG). Thus BMGs represent theoretical terms whose evaluation leads in one step to their observational equivalents. We determine BMGs by stratification, that means we determine first all BMGs from the original theory, reduce the theory by the corresponding clauses and proceed to determine the BMGs for the next higher level.

A plausible algorithm for validation by reconstruction contains then the following steps:

- Determine the set of stratified BMGs from the given theory (see figure 4).
- For every BMG determine all GPMs by partial evaluation (see figure 3).
- Build for every GPM a rule whose head is given by the BMG.
- Invoke a "reconstruction" algorithm (e.g. inverse resolution) with the generated rule.
- Terminate, if the original theory was reconstructed, else generate the next rule.
- If no further rule can be generated then terminate.

```
lighter(X,Y) :- weight(X,W1), weight(Y,W2), W1 < W2.
weight(X,W)  :- object(X), volume(X,U), density(X,V), W is U*V.
weight(X,5)  :- endtable(X).
object(X)    :- endtable(X).
object(X)    :- box(X).
```

Figure 5: "Lighter" Theory

```
Level 1: object(A) :- box(A)
         object(A) :- endtable(A)
Level 2: weight(A,5) :- box(A), volume(A,B), density(A,C), 5 is B*C.
         weight(A,5) :- endtable(A).
Level 3: lighter(A,E) :- box(A), volume(A,B), density(A,C), D is B*C,
                         box(E), volume(E,F), density(E,G), H is F*G,
                         D < H.
         lighter(A,E) :- box(A), volume(A,B), density(A,C), D is B*C,
                         endtable(E), D < 5.
         lighter(A,B) :- box(B), volume(B,C), density(B,D), E is C*D,
                         endtable(A), 5 < E.
```

Figure 6: Constructed Rules

This approach has the advantage that the user intention can be handled in an implicit way. Questions which are posed to the user in the reconstruction phase can be viewed as experiments. If the user has a clear idea of the intended system, then it is possible to reconstruct the theory on the basis of his answers. Thus all questions which lead to the reconstruction of the theory must be answered positively by the user. If the user answers these questions negatively, a different theory is "re"-constructed. Thus we can conclude that the user intention has changed.

By this approach we can avoid the following problems determined in the last section:

1. We avoid to acquire an interpretation function with an infinite number of models extensionally.

2. We perform experiments about theoretical terms implicitly, if a predicate is reinvented in the reconstruction phase and it is possible to reconstruct the original theory from the user's answers.

3. We need not to avoid infinite branches, thus we can handle recursive theories. The termination criterion assures that we can stop the generation of GPMs as soon as the original theory was reconstructed.

# 7 An Exemplifying Experiment

Until now determination of stratified BMGs, partial evaluation and inverse resolution are still isolated modules. We have performed a small experiment to exemplify the approach of validation by reconstruction. For that purpose we have taken a part of a small, well-known theory (adopted from Mitchell et al. [12]).

We computed the stratified bottommost goals, from the theory shown in figure 5, and constructed the corresponding rules shown in figure 6. These rules were used afterwards as input to our experimental inverse resolution system to reconstruct the original theory. In the following we discuss some of the interactions between the user and the system, to give a flavor of the reconstruction process.

```
⟹ weight(_439,5) :- box(_439), volume(_439,_445),
     density(_439,_449), 5 is _445*_449.
Rule weight(A,5) :- box(A), volume(A,B), density(A,C), 5 is B*C
     added to theory.
ABSORPTION found rule: object(A) :- weight(A,B).
Is this rule always true ? no.
                                    ⋮
```

Figure 7: Overgeneral knowledge avoided

```
⟹ weight(_439,5) :- endtable(_439).
Rule weight(A,5) :- endtable(A) added to theory.
ABSORPTION found rule: weight(A,5) :- object(A).
Is this rule always true ? no.
IDENTIFICATION found rule:
     endtable(A) :- object(A), volume(A,B), density(A,C), D is B*C.
Is this rule always true ? no.
INTRA-CONSTRUCTION invents predicate:
     PRED(A,B) :- endtable(B).
     PRED(A,B) :- object(B), volume(B,C), density(B,D), A is B*C.
     weight(A,5) :- PRED(5,A).
Are these rules always true ? no.
                                    ⋮
```

Figure 8: Correct rejections w.r.t. the user intention

In figure 7 the absorption operator has proposed a clause which falls under the 'case of the missing argument'. The proposed clause is overgeneral, because a qualifying predicate is missing (i.e. the variable B can be instantiated with any individual of the domain). The user recognized this and rejected the proposed clause.

In figure 8 the user rejected the clause proposed by the absorption operator, because in his interpretation only endtables can have the weight 5. The second clause proposed by the identification operator was also rejected, because the interpretation of this clause made no sense to the user. In the third proposal the intra-construction operator invented the 'case of the argument order', which was also rejected by the user.

The absorption operation in figure 9, produced a rule which the user accepted, because not only boxes can be lighter than endtables.

Because our inverse resolution system is still in an experimental development stage and uses an exhaustive breadth-first strategy to find all possible operator applications the user

```
⟹ lighter(_439,_440) :- box(_439), volume(_439,_441),
     density(_439,_442), _443 is _441*_442, endtable(_440), _443 < 5.
Rule lighter(A,B) :- box(A), volume(A,C), density(A,D), E is C*D,
     endtable(B), E < 5 added to theory.
ABSORPTION found rule: lighter(A,B) :- object(A), volume(A,C),
     density(A,D), E is C*D, endtable(B), E < 5.
Is this rule always true ? yes.
                                    ⋮
```

Figure 9: Acceptable under the user intention

had to answer 31 questions, 24 questions were answered negatively, 7 positively, and the system reinvented the theoretical term 'weight' before the original theory was reconstructed.

Of course, the number of questions posed in the reconstruction process seems to be a major impediment for practical applications. As Muggleton [15] has shown the number of questions posed by an inverse resolution system can be reduced by an information theoretical threshold measure. This measure computes the 'theory compression' that can be achieved by an inverse resolution operator application. Whether our approach can be successfully extended by such a measure will be discussed elsewhere.

# Conclusion

Knowledge validation, in analogy to knowledge acquisition, can be viewed under two assumptions. One assumption, that the user has a fixed idea of the intended system, corresponds to the 'bottleneck' view of knowledge acquisition. The other assumption, that also the user's idea of the intended system is evolving, corresponds to the 'sloppy modeling' view.

Most constructive induction techniques implictly assume a perfect user, who not only makes no errors, but also does not change his interpretation. Of course, users are in general not perfect. Thus we have introduced the notion of user intention. This definition of an interpretation function seems on one hand general enough to capture and handle changing interpretations. On the other hand it allows us to integrate already existing results and systems based on fixed interpretation functions into incremental knowledge validation systems.

We discussed the problems that arise when we try to acquire the user intention by experimentation extensionally. The main problems: acquisition by experimentation is extensional, validation of theoretical predicates is not possible, and recursive theories cannot be handled, can be avoided by a modified validation technique called validation by reconstruction. This technique is based on inverse resolution and benefits from "natural traps" which are frequently produced by inverse resolution. Although this approach is not yet totally integrated and more empirical evidence for it is needed, an exemplifying example indicated its realizability.

# References


[1] Benbasat, I., Dhaliwal, J.S., The Validation of Knowledge Acquisition: Methodology and Techniques, *Proceedings of the EKAW-89*, 1989.

[2] Genesereth, M.R., Nilsson, N., *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann, Los Altos, California, 1987.

[3] Ginsberg, A., Knowledge-Base Reduction: A New Approach to Checking Knowledge Bases for Inconsistency & Redundancy, *Proceedings of the AAAI-88*, 1988.

[4] Ginsberg, A., Knowledge Base Refinement and Theory Revision, *Proceedings of the 6th International Workshop on Machine Learning*, 1989.

[5] v. Harmelen, F., Bundy, A., Explanation-based Generalisation = Partial Evaluation, Research Note, *Artificial Intelligence 36*, 1988.

[6] Hempel, C.G., *Philosophie der Naturwissenschaften*, Wissenschaftliche Reihe 4144, dtv, München, 1974, translation of: *Philosophy of Natural Science*, Foundations of Philosophy Series, Prentice-Hall, Englewood Cliffs, 1966.

[7] Hoppe, Th., Hypotheses Generation for Knowledge Validation, to appear in: *Proceedings of the ECAI-90*, 1990.

[8] Lloyd, J.W., *Foundations of Logic Programming*, second extended edition, Springer-Verlag, Berlin, Heidelberg, New York, 1987.

[9] López, B., Meseguer, P., Plaza, E., Validation of Knowledge Based Systems: A State of the Art, Report de Recerca GRIAL 89/9, Centre d'Estudis Avançats de Blanes, 1989.

[10] de Mántaras, R.L., Kerdiles, P.F., Larsen, H., Validation Methods and Tools for Knowledge-Based Systems (VALID), in: *Wissensbasierte Systeme*, W. Brauer, C. Freksa (eds), Proceedings of the 3rd International GI-Kongress, München, Oktober 1989, Informatik-Fachberichte 227, Springer-Verlag, Berlin, Heidelberg, New York, 1989.

[11] Meseguer, P., A New Method to Checking Rule Bases for Inconsistency: A Petri Net Approach, Draft, submitted to ECAI-90.

[12] Mitchell, T.M., Keller, R.M., Kedar-Cabelli, S.T., Explanation-based Generalisation: A Unifying View, in: *Machine Learning I*, Kluwer Academic Publishers, Boston, 1986.

[13] Morik, K., Sloppy Modeling, in: *Knowledge Representation and Organization in Machine Learning*, K. Morik (ed), Lecture Notes in Artificial Intelligence 347, Springer-Verlag, Berlin, Heidelberg, New York, 1989.

[14] Muggleton, S., Duce, an oracle based approach to constructive induction, *Proceedings of the 10th IJCAI-87*, Morgan Kaufmann, Los Altos, California, 1987.

[15] Muggleton, S., A strategy for constructing new predicates in first order logic, *Proceedings of the 3rd European Working Session on Machine Learning*, Pitman, London, 1988.

[16] Muggleton, S., Buntine, W., Machine Invention of First-order Predicates by Inverting Resolution, *Proceedings of the 5th International Workshop on Machine Learning*, Morgan Kaufmann, San Mateo, 1988.

[17] O'Keefe, R.M., Balci, O., Smith, E.P., Validating Expert System Performance, *IEEE Expert Winter*, 1987.

[18] de Raeth, L., Bruynooghe, M., Constructive Induction by Analogy: a method to learn how to learn?, in: *Proceedings of the 4th European Working Session on Machine Learning*, K. Morik (ed), Montpellier, December 1989, Pitman, London, 1990.

[19] Rousset, M.-Ch., On the Consistency of Knowledge Bases: The COVADIS System, *Proceedings of the ECAI-88*, 1988.

[20] Rouveirol, C., Puget, J.-F., A Simple Solution For Inverting Resolution, in: *Proceedings of the 4th European Working Session on Machine Learning*, K. Morik (ed), Montpellier, December 1989, Pitman, London, 1990.

[21] Shapiro, E.Y., Inductive Inference of Theories From Facts, Research Report 192, Yale University, Department of Computer Science, 1981.

[22] Shapiro, E.Y., Algorithmic Program Debugging, Research Report 237, Yale University, Department of Computer Science, 1982, MIT-Press, Cambridge, 1983.

[23] Shaw, M.L.G., Woodward, J.B., Validation in a knowledge support system: construing and consistency with multiple experts, *International Journal of Man-Machine Studies 29*, 1988.

[24] Sterling, L., Shapiro, E., *The Art of Prolog*, MIT-Press, Cambridge, Massachusetts, London, England, 1986.

[25] Suwa, M., Scott, A.C., Shortliffe, E.H., Completeness and Consistency in a Rule-based System, in: *Rule-based Expert Systems*, B.G. Buchanan, E.H. Shortliffe (eds), Addison-Wesley, 1985.

[26] Wirth, R., Lernverfahren zur Vervollständigung von Hornklauselmengen durch inverse Resolution, Ph. D. thesis, University of Stuttgart, 1989.

[27] Wirth, R., Completing Logic Programs by Inverse Resolution, in: *Proceedings of the 4th European Working Session on Machine Learning*, K. Morik (ed), Montpellier, December 1989, Pitman, London, 1990.

[28] Wrobel, S., Demand-driven concept formation, in: *Knowledge Representation and Organization in Machine Learning*, K. Morik (ed), Lecture Notes in Artificial Intelligence 347, Springer-Verlag, Berlin, Heidelberg, New York, 1989.

# Models of Problem-Solving: One label - One idea ?

Werner Karbach, Marc Linster and Angi Voß

Expert System Research Group
GMD
PO Box 1240
D-5205 St. Augustin 1

e-mail: karbach@gmdzi.uucp, linster@gmdzi.uucp, avoss@gmdzi.uucp

We compare different approaches to the modeling of problem solving processes in knowledge based systems to find out how they model knowledge, how they represent it, how flexible they are and to what degree they guide or support the development of model-based systems. We focus on Generic Tasks, KADS, role limiting methods and the method-to-task approach.
For each approach we describe key concepts, motivations, premises, claims and achievements. These are used to compare the terminology of the approaches, and to establish a mediating terminology for the rest of the paper. The prominent models are compared to show which problem areas they cover, and what methods they use.
We compare the generic and the task-dependent knowledge to explicate the knowledge needs of the different models. Finally we point out fundamental, but hopefully complementary differences between the approaches.

## 1. Introduction

The term model has been widely used in the construction of knowledge based systems. A model is a purposeful abstraction that allows us to reduce complexity by focussing on certain aspects. Researchers in AI have examined models to represent qualitative process relations [11], qualitative dynamics [15], temporal relations [1, 25], domains [14] and problem solving [3, 20, 6, 18, 12]. For our purpose we will focus on the models of problem solving and their role in the development of knowledge based systems.

All model based approaches to problem solving rely on the following hypotheses:

(1) It is useful to describe problem solving behavior on a more adequate abstract level than is offered by general purpose knowledge representation languages. These high-level descriptions are called models of problem-solving;

(2) Models of problem solving behavior can be specified in a problem specific, but application-task independent way;

(3) Models guide the knowledge acquisition process and structure the knowledge base.

To illustrate the general direction research is taking and, at the same time illustrate the fundamental and terminological differences between the approaches that make up the state of the art, we picked four advocates: Generic Tasks [5, 6, 7], KADS [24, 3, 4], the method-to-task approach and the role-limiting-method approach. The label "method-to-task" was chosen for the approach taken in

the PROTÉGÉ/p-OPAL combination [20, 21], and we hope that "role-limiting-method" approach is a good characterization of the modeling technique underlying SALT, MOLE or KNACK [16, 17, 10, 13, 18]. We chose labels for the approaches rather than using names of systems, as we think that the different systems are only instantiations of the approaches and do not render their full potential. As most approaches have been developed by teams of researchers, names of authors would not do either.

For our comparison of the approaches we synthesized information from the different papers in the bibliography. In general we will not give any more explicit references. Assuming some familiarity with the approaches discussed, we will only give short surveys of the individual approaches: their key concepts, premises, claims and achievements. With this preparation, we can compare basic terms like *task* or *conceptual model* that recur with most authors. We will see that there are more or less significant deviations in the way they are used. We will establish our terminology for this paper by attempting a synthesis.

Moving on a more solid terminological ground, we will then present the concrete models developed by the four approaches and compare them with respect to the problems they tackle and the problem solving methods they use. We will describe which parts of the models are generic, what kind of knowledge must be added and by whom, whether the models are operational, if they can be combined or if there is a modeling language that allows the development of new models. This comparison is done to reveal common features, expose differences and to discuss the three hypotheses stated above, thus hopefully allowing for a future synthesis.

## 2. Survey of Four Major Approaches

We give short surveys of the Generic Task, KADS, method-to-task and role-limiting-method approaches to the model based development of a knowledge based system. For reasons of simplicity we use the following abbreviations: GT (Generic Tasks), MT (method-to-task) and RLM (role-limiting method). To prevent the implication of any judgement, we will present them in alphabetical order of the abbreviations. For each, we summarize the key concepts, motives, premises, claims and achievements. With *key concepts* we mean characteristic, central notions, the *premises* represent working hypotheses and underlying assumptions, and *claims* comprise advantages, purposes or goals of the approach. Under *achievements* we sum up what has been accomplished so far.

### 2.1. The Generic Task Approach

Papers underlying the presentation: [6, 7, 8, 5] by Brown, Bylander and Chandrasekaran.

#### 2.1.1. Key Concepts

***interaction problem:***
Knowledge cannot be represented free of assumptions of how it will be used, e.g. the form and the representation in which knowledge is needed for design is different from the one needed for diagnosis.

***generic task:***
A generic task is characterized by its input and output. It is a way to organize and represent the knowledge needed to solve certain kinds of problems, including a vocabulary of knowledge types, and the process (algorithm, control, problem solving) used as a problem solver.

#### 2.1.2. Motives

Separating knowledge and inference, as is done in universal languages, leads to systems which are at too low a level of abstraction, causing task level phenomena to be lost in details and raising

implementation level phenomena to a theoretical status. Such systems are either inefficient or encode control aspects in the knowledge base.

### 2.1.3. Premises

The interaction hypothesis states that knowledge cannot be represented free of assumptions of how it will be used. This interaction problem can be exploited to select portions of domain knowledge and encode them into an efficient and maintainable problem solving structure, i.e. the generic task.

### 2.1.4. Claims

Building blocks of reasoning strategies can be identified that are generic and widely useful as components of complex reasoning tasks. With the identification of generic tasks languages can be developed that encode both the problem solving strategy and the knowledge that is appropriate for solving problems of that type. These languages facilitate the development by giving the knowledge engineer access to tools which work closer to the level of the problem, not on the level of implementation languages such as rules or frames.

### 2.1.5. Achievements

So far, several generic tasks have been identified and several languages were developed, in particular for the definition of diagnostic and routine design tasks.

## 2.2. The KADS Approach

Papers underlying the presentation: [3, 4, 24, 23] by Breedeweg, Breuker, Davoodi, Wielinga and Schreiber.

### 2.2.1. Key Concepts

***domain layer:***
The static knowledge of the domain is described in terms of *concepts* and *relations*.

***inference layer:***
The inference layer specifies the inference making competence that is required to solve a problem using the domain knowledge. It describes what inferences can be made, but not how or when they are made. It consists of meta-classes and knowledge sources which together constitute the *inference structure*. A *knowledge source* is a primitive inference making function using knowledge provided by domain relations. Its input and output parameters are *meta-classes*, which describe roles that domain concepts can play in a problem solving process.

***task layer:***
It specifies the task decomposition used to control the inferences in terms of tasks and goals. A *task* combines knowledge sources and subtasks in order to achieve a particular goal. A *fixed task structure* realizes a particular strategy to solve a range of problems.

***strategic layer:***
On this layer plans the execution of the tasks is controlled and monitored, and impasses of the problem solving process are diagnosed and repaired. This layer may be absent if the range of problems can be solved by using a same strategy (*fixed task structure*), and if the interaction between communication tasks and problem solving tasks is simple.

***conceptual model:***
This is a term borrowed from Norman [22] that denotes the model the designer of a computer system has in mind and/or specifies when implementing a system. In KADS it models the expertise by structuring the required knowledge, inference competence and use of knowledge. KADS a semi-formal *modelling language* for describing conceptual models on four different layers corresponding to the different types of knowledge: domain, inference, task and strategic layer.

***interpretation model:***
This is a model of the problem solving process in a class of tasks over a type of domains. It provides an initial structure to the data and allows for a mainly top-down data analysis process. The interpretation model comprises strategic, task and inference layer. That means it is a conceptual model without domain layer. Elementary interpretation models that are used frequently are called *generic interpretation models.*

***design model:***
A design model describes the transformation of a conceptual model into an implementation on the basis of external and user requirements. It consists of three layers describing resp. the function, the behavior and the structure of the artifact.

### 2.2.2. Motives

Rapid prototyping approaches are inadequate for the systematic development of large knowledge based systems as they do not strictly separate the analysis from the design and implementation phases. Developing a KBS is essentially a modelling activity. It is a mapping from an understanding of behavior in the real world to the description in form of an artifact: an engineering construct. The analysis of knowledge should be model driven as early as possible, as this makes the process more efficient and more proficient.

### 2.2.3. Premises

Knowledge and expertise can be analyzed before design and implementation starts. It is possible to sail between the Scylla of natural language and the Charibdis of a formal language, and express a model of expert problem solving at the epistemological level. It is useful to distinguish several generic types of knowledge (domain, inference, task and strategic), corresponding to different roles that knowledge can play in reasoning processes. These types of knowledge can be organized in layers, which have only limited interaction.

### 2.2.4. Claims

Control knowledge can be specified in a domain independent way. It is possible to identify constraints and top-down refinement models to guide the knowledge acquisition process in the framework of a comprehensive knowledge engineering methodology with a life cycle model, a modelling language, tools and techniques.

### 2.2.5. Achievements

The KADS group developed layered modelling languages for conceptual models and design models. In their approach, a conceptual model can be built from scratch or by selecting interpretation models, combining them and adding the domain layer. KADS has a library of interpretation models from which the knowledge engineer can choose.

## 2.3. The Method-to-Task Approach

Papers underlying the presentation: [19, 20, 21] by Musen, Fagan, Combs and Shortliffe.

### 2.3.1. Key Concepts

***conceptual model***
The conceptual model provides a set of terms and relations that define the semantics of the knowledge-base elements that the program displays, and that ascribe meaning to the data entered by the user.

***representation-based conceptual model***
TEIRESIAS-like tools allow the entry of knowledge in basic knowledge-representation formalisms, like frames, rules or clauses.

***method-based conceptual model***
Knowledge is entered in the terminology of the underlying problem-solver, i.e. as observations, fixes, requirements.

***task-based conceptual model***
Knowledge is entered in application-specific terms, e.g. as illness, drug dose, blood cell counts.

***content knowledge***
Content knowledge is domain knowledge that is used by a problem solver in the way that is predetermined by the problem-solving process.

### 2.3.2. Motivation

The development of knowledge-based systems should be facilitated by separating the task of the knowledge engineer from the task of the domain-expert, each doing what he is best at.

### 2.3.3. Premises

Problem-solving knowledge can be modeled separately from the content knowledge, and domain-knowledge requirements (i.e. the needed content knowledge) can be described in terms of types that must be instantiated.

### 2.3.4. Claims

In domains that involve a number of similar tasks it is possible to create a systematic domain terminology and a general model of the building blocks of the tasks of the domain, i.e. a task-based conceptual model. These can be instantiated for the various tasks. Thus the difficult part of the knowledge acquisition process (defining the inventory and the problem-solving terminology) need not be repeated for every knowledge-base in the same domain.

Knowledge acquisition tools cannot help get rid of the knowledge engineer.

### 2.3.5. Achievements

The MT knowledge acquisition methodology distinguishes tools that define method-based conceptual models, and tools that specialize them to task-based conceptual models. PROTÉGÉ and p-OPAL are representatives. For applications in the domain of cancer therapy PROTÉGÉ defines the terminology and the relations needed to describe treatment protocols for cancer therapies; p-OPAL instantiates this knowledge for specific therapies. Thus the knowledge acquisition process is separated in two phases with separate responsibilities for the knowledge engineer and the domain-expert.

## 2.4. The Role-Limiting-Method Approach

Papers underlying the presentation: [10, 13, 16, 17, 18] by Eshelman, Klinker, Marcus and McDermott.

### 2.4.1. Key Concepts

***problem solving:***
Problem solving is the identification, selection and implementation of a sequence of actions that accomplish a task within a specific domain.

***problem-solving method:***
A problem solving method provides a means of identifying, at each step, candidate actions. It provides one or more mechanisms for selecting among candidate actions and ensures that the selected action is implemented. In the RLM approach a problem solving method is close to what traditionally was called an inference engine.

***weak method:***
This is a term borrowed from Newell. A weak method is a problem-solving method that is only weakly constrained by task features; each method is potentially applicable to a broad set of task types. Weak methods typically achieve this generality with simple control structures that require additional control knowledge to be supplied before the method can be applied. They cannot provide much help in determining what knowledge needs to be collected or how it should be encoded.

***role-limiting method:***
A role limiting method can be viewed as a specialization of a weak method. It guides knowledge collection and encoding and predefines the structure of the task-related control knowledge the method can use. A role-limiting method typically consists of a simple loop over a sequence of 5 to 10 steps. Some of the steps operate on significant bodies of task specific knowledge. Within a step there is no control, that is, it makes no difference in what order the actions are performed.The role-limiting-method itself has little, if any, task specific knowledge.

### 2.4.2. Motivation

There are no good descriptions of the various problem solving methods expert systems use and no characteristics of appropriate tasks.

### 2.4.3. Premises

Many, but probably not all tasks can be performed with modest amounts of task specific control knowledge.

### 2.4.4. Claims

Role-limiting methods have a broad scope of applicability and they provide substantial help in specifying what knowledge needs to be acquired to perform a task and how this knowledge can be appropriately encoded.

### 2.4.5. Achievements

Discriminating features of problem solving methods used in expert systems have been identified. Tools have been built that tackle classes of tasks, each of the tools knowing what sort of knowledge is required by the underlying problem solver.

## 2.5. Comparison of Some Basic Common Concepts

The concepts of *task, problem solving method, implementation level*, a "more abstract level", *control knowledge* and *conceptual model* are basic to all four approaches, though they may not always be explicitly mentioned or occur under different labels.

Table 1: Comparison of the terms used by the authors of the different approaches. Key concepts are in italics.

| Term | GT approach | KADS | MT approach | RLM approach |
|---|---|---|---|---|
| Task | Application of problem solving methods independently of a particular domain. | Colloquial use: application of problem solving methods to problems in a particular domain.<br>Formal use: a task is a control construct on the *task layer* of the KADS *interpretation* and *conceptual models*. | Application of problem solving methods to problems in a particular domain.<br>The reason to write an expert system. | Application of problem solving methods to problems in a particular domain. |
| Problem solving method | Here the terms "inference strategy" and "(*generic*) *task*"are used. The latter determines the knowledge representation (cf. *interaction problem*) | *Interpretation models* are descriptions of problem solving methods. | A domain independent description of how to reach a solution. | A domain independent description of how to reach a solution that it is expressed as a sequence of "identify, select and implement" steps. |
| Modeling level(s) | On the task level the generic tasks are formulated as a problem solving method with a knowledge representation. | Brachmann's epistemological level [2] is the level of development for the conceptual models. The models themselves consist of descriptions on four levels: domain, inference, task and strategy. | Two levels are distinguished: (1) development of a *task model* by the knowledge engineer at the PROTÉGÉ level; (2) instantiation of the *task model* by the expert at the p-OPAL level. | ---------- |
| Implementation level | Rule/frame/logic level | The implementational level is the level where the knowledge based systems are programmed. This can be Prolog, a shell or any other language. | ---------- | ---------- |
| Control knowledge | ---------- | On the *task layer* control is specified in a domain independent way. | Control and control knowledge is formulated in domain terms. | "... a method's control knowledge comprises (1) an algorithm specifying when to use what knowledge to identify, select and implement actions together with (2) whatever knowledge the method has for selecting among candidate actions." |
| Conceptual model | ---------- | Colloquial use: the model of expertise that the knowledge engineer has in mind.<br>Formal use: description of knowledge on four layers that will be implemented via the *design model*. | The *conceptual model* provides a set of terms and relations that define the semantics of the knowledge base elements that the program displays, and that ascribe meaning to the data entered by the user. | ---------- |

To establish the terminology of this paper, we attempt the following synthesis.

***task:***

We prefer the meaning chosen in the MT and RLM approaches, which also coincides with the colloquial use in KADS, i.e. a task is the application of a problem-solving method to a problem in a particular domain. Examples are the application of skeletal-plan refinement to the development of individual treatment protocols in the domain of cancer therapy, or the application of hierarchical classification to fault-finding in four-stroke four cylinder engines of a special manufacturer.

***problem solving method:***

Our intuitive understanding of the term coincides with the definition used in the MT approach, as it is more general than the one used in the RLM, GT or KADS approach. Generic tasks and interpretation models are special ways to describe problem solving methods.When trying to characterize problem solving methods by their control and control knowledge we saw the need to be more precise, as control is too fuzzy a notion. Extending the RLM definition we adopted the following meaning:

A *problem solving method* is characterized by its types of actions, the selection between the types, the selection between the instances of the selected type and the execution of the selected action. The basic procedure consists of: (1) recognizing what type of action comes next; (2) which particular actions are available of that kind; (3) select one of those and (4) execute it.

*Actions* are the elementary steps executed by the method. An action may be completely task-neutral, or completely task-specific, or it may have some generic and some task-specific aspects.

Problem solving methods operate on data structures. The *data structures* manipulated by the actions are usually defined by the problem solving method, but sometimes they may be refined in a task-specific way, while the concrete *data* to be filled in are usually completely task-specific. If the next action to be executed cannot be determined in a task-independent way, task-dependent *selection criteria* must be provided.

***modeling level:***

The modeling level provides a problem solving specific language which may, but need not be operational.

***implementation level:***

This is the level of general purpose languages that are not specialized to any problem solving methods. The borderline between modelling and implementation levels may not always be easy to draw.

***conceptual model:***

This is a model of a system builder about how knowledge is used by a person or a computer program and which roles it plays in that process. That means, conceptual models may be mental as well as verbalized, and they may model the expert as well as the system or a completely new approach invented by the knowledge engineer.

## 3. Comparison of the models

Each of the four approaches has elaborated several substantial problem-solving models. We will give a short characterization of the most well known ones before we cluster the models according to the problem class they tackle and to their problem solving methods.

## 3.1. Description of the major models

We present the most prominent problem-solving models of the four approaches according to a common scheme: *Problem class* specifies the major class of problems which can be tackled. We only distinguish analysis and synthesis. See section 3.2 for a definition. We do not use a finer distinction because we do not want to address the whole issue of problem type classification in this paper. The original classification by the respective authors is included in parentheses. *Method* gives the name of the problem solving method used and *description* a short verbal description thereof. *Knowledge needs* describe the information needed to supplement the problem solving method for a particular task. They specify the view on the world required to apply the models.

### 3.1.1. Generic Tasks

***CSRL:***

| | |
|---|---|
| Problem class: | analysis (hierarchical classification) |
| Method: | establish-refine |
| Description: | If a node in the hierarchy is successful consider its children. If a node fails, reject it and prune the subtree. |
| Knowledge needs: | classification hierarchy, matching from features to concepts |

***DSPL:***

| | |
|---|---|
| Problem class: | synthesis (routine design) |
| Method: | plan refinement, skeletal planning |
| Description: | A predefined design plan is top-down refined. Knowledge for backtracking is available. |
| Knowledge needs: | hierarchical structure of the object to be designed, design plans for each node in the hierarchy, backtracking knowledge |

***HYPER:***

| | |
|---|---|
| Problem class: | analysis (hypothesis-matching) |
| Method: | matching |
| Description: | In a hierarchy of abstractions evidences are propagated bottom-up. |
| Knowledge needs: | hierarchy of feature abstractions, procedures to determine the evidence of high level abstractions |

***IDABLE:***

| | |
|---|---|
| Problem class: | analysis (knowledge-directed-information-passing) |
| Method: | abstraction |
| Description: | Knowledge is generated using inference procedures or default knowledge. |
| Knowledge needs: | data concepts with default values, methods for infering data |

***PEIRCE:***

| | |
|---|---|
| Problem class: | synthesis (abductive hypothesis assembly) |
| Method: | cover-and-differentiate |
| Description: | Searches for a set of hypotheses which best explain a given situation. |
| Knowledge needs: | hypotheses, causal relationships between hypotheses, plausibility values |

### 3.1.2. KADS' Interpretation Models

***Systematic diagnosis: localization, causal tracing***

| | |
|---|---|
| Problem class: | analysis (diagnosis) |
| Method: | A system model is decomposed (component-wise resp. causally). Complaints are compared with standard values in each decomposition to identify faulty components resp. faulty states. |
| Knowledge needs: | system model, variable value, norm, difference, conclusion |

***Heuristic classification***

| | |
|---|---|
| Problem class: | analysis (diagnosis) |
| Method: | The method is based on the "horse-shoe model" of Clancey [9]. Observables are abstracted to reach a solution class via matching. This solution class is further refined into a solution. |
| Knowledge needs: | observables, variables, solution abstractions, solutions |

***Assessment***

| | |
|---|---|
| Problem class: | analysis (assessment) |
| Method: | A case description is compared to a system model. Via abstraction and matching the discrepancy between both is infered and classified into a decision class. |
| Knowledge needs: | case description, abstract case description, norm, discrepancy, decision classes |

***Monitoring***

| | |
|---|---|
| Problem class: | analysis (monitoring) |
| Method: | From a system model a parameter set is selected to be monitored. Observables are compared to the parameters and in the case of discrepancies classified into a discrepancy class. |
| Knowledge needs: | observables, variable values, system model, parameter values, differences, discrepancy classes |

***Prediction***

| | |
|---|---|
| Problem class: | analysis (prediction) |
| Method/Description: | From a description of the processes and objects of a system and the knowledge about their behavior the future state of the system is predicted (e.g. by default reasoning, simulation or constraint propagation. |
| Knowledge needs: | augmented system description (= system description + system model), predicted system description |

***Prediction of behavior***

| | |
|---|---|
| Problem class: | analysis (prediction) |
| Method/Description: | The state of the system is predicted based on a causal system model. |
| Knowledge needs: | system model, state description, augmented state description, parameter values, predicted state description |

***Design based on given abstract models (a general description)***

| | |
|---|---|
| Problem class: | synthesis (design) |
| Method/Description: | From informally specified requirements a formal description is infered. An abstract model of the object to be designed guides the construction of the artefact. |
| Knowledge needs: | informal problem statement, formal specification, conceptual model of the final product, detailed design |

***Hierarchical design***

| | |
|---|---|
| Problem class: | synthesis (design) |
| Method/Description: | The abstract model is decomposed into components. Each component is designed separately. The detailed component designs are then aggregated. |
| Knowledge needs: | formal specification, skeletal model, components, model of component interaction, detailed model |

***Incremental design (only inference structure given)***

| | |
|---|---|
| Problem class: | synthesis (design) |
| Method/Description: | From a full specification given design elements can be chosen by transformation rules. They are composed using a skeletal model. |
| Knowledge needs: | conceptual model of the detailed design, functions design elements, model class, skeletal model, detailed design |

### 3.1.3. Method-to-Task Approach

***PROTÉGÉ:***

| | |
|---|---|
| Problem class: | synthesis (planning) |
| Method: | skeletal-plan-refinement |
| Description: | The method assumes a basic plan outline to exist so that the system can concentrate on filling in the details. |
| Knowledge needs: | Planning entities constitute a skeletal plan (classes, attributes, attribute properties).Task-level actions can modify the manner in which the planning entities are carried out. Input data is categorized in classes. |

### 3.1.4. Role-Limiting-Method Approach

***MOLE-p:***

| | |
|---|---|
| Problem class: | analysis (diagnosis) |
| Method: | cover-and-differentiate |
| Description: | Candidate hypotheses are activated to explain complaints. Further observations are requested in order to differentiate between hypotheses. |
| Knowledge needs: | states (complaints, hypotheses), causal relation between states |

***SALT-p:***

| | |
|---|---|
| Problem class: | synthesis (construction) |
| Method: | propose-and-revise |
| Description: | Propose values of design parameters and check constraints on the values. If a constraint is violated apply fixes to revise the value by applying remedies. |
| Knowledge needs: | value-dependency network, value constraints, fixes |

***SIZZLE-p:***

| | |
|---|---|
| Problem class: | |
| Method: | Case-based reasoning |
| Description: | An appropriate case is chosen from a case library based on a similarity metric. There exists predefined knowledge how to adapt a case to the new problem. |
| Knowledge needs: | similarity between cases, extrapolation knowledge |

***YAKA-p:***

| | |
|---|---|
| Problem class: | analysis (diagnosis) |
| Method: | qualitative reasoning |
| Description: | Abnormal values are identified with a model of normal system behavior. Traces over qualitative equations are used to identify a set potential causes. This set is differentiated by a cover-and-differentiate method. |
| Knowledge needs: | model of normal functioning system |

## 3.2. Scope of the Models

We will now classify the models according to their scope, i.e. the problem class they are concerned with and the problem solving method they employ. To draw the distinction between analysis and synthesis, we look at the results of the implemented problem-solving methods. If the method selects from a predefined set of solutions, then it is an analysis type. If it constructs its solutions, then it is a synthesis type. Obviously this distinction is a definitional one, maybe it is artificial. MOLE can be viewed as a system that selects its solution, or as one that synthesizes an explanation path. Nevertheless it seems beneficial to consider the distinction between synthetic and analytic problem solvers.

All models for synthesis are based on a predefined structure of the artefact to be designed. *PROTÉGÉ*, *DSPL* and *Hierarchical design* use skeletal-plan refinement as problem solving method. *SALT* proposes design extensions using a data-dependency network. *Incremental design* can also be seen to fall into this category. For problem solving methods in analytic problems, *CSRL* and *Heuristic classification* stand for an associative approach while *MOLE*, *PEIRCE* and *Systematic diagnosis* operate on causal networks. *Monitoring* and *Prediction* also fit into the analytic category.

*SIZZLE*, *YAKA* and *Prediction of behavior* are representatives of non-standard approaches to problem solving. Conventional problem solving methods tackle problems from scratch using shallow knowledge. *SIZZLE* allows the use of experience in problem solving by case based reasoning. *YAKA* and *Prediction of behavior* integrate model-based reasoning based on deep knowledge. These three models can be seen as complementary to the problem solving methods in the previous paragraph.

Some of the proposed models, like *IDABLE*, *HYPER* and *Assessment* represent techniques widely used in AI. They are not related to a special task or problem solving method. For example, data abstraction -like in *IDABLE*- can be used in diagnosis to abstract data to features. In synthesis problems it can be used for abstracting features of a complete system on the basis of the features of each single component. These models can be seen as basic ingredients for problem solving methods. They are comparable to standard knowledge sources in KADS, although they can expand to a complete model like in *Assessment*.

## 4. Comparison of the Approaches

The models presented in the previous section specify problem solving methods on the modelling level. Thus they describe, according to hypothesis (I) of the introduction, a "problem solving behavior on a more abstract level than is offered by general purpose programming languages". As

claimed there, these descriptions are useful, because they are closer to the task and ease the communication between expert and knowledge engineer.

To assess hypothesis (II), i.e. "models of problem solving behavior can be specified in a problem specific, but application-task independent way", we will now analyze what knowledge is specified in a task-independent resp. task-dependent way. When establishing our use of the term "problem-solving-method" in section 2, we took pains to identify potentially task-specific aspects of a problem-solving-method. Correspondingly, in table 2 we separate the (kernel of the) *problem solving method* from its actions, data and *selection criteria*. We distinguish generic from instantiated actions and data structures from concrete data. *Generic actions* and *data structures* describe the format of *instantiated actions and concrete data*, respectively. However, we were unable to distinguish generic actions from the data structures, nor instantiated actions from the concrete data they operate on. This is partly due to poor descriptions in the literature and to the fact, that in object-oriented languages the distinctions between data structures and action descriptions are muddled.

Table 2: How different types of knowledge are handled in the models of the four approaches.

| Types of knowledge | GT approach | KADS | MT approach | RLM approach |
|---|---|---|---|---|
| Problem solving method | Several predefined languages with interpreters. | Interpretation models. | Predefined e-ONCOCIN interpreter. | Several predefined interpreters. |
| Generic Actions and the data structures they operate on | Each language provides a set of generic actions and the data structures they operate on. | Defined as knowledge sources and meta-classes on the inference layer. | Planning entities are refined into domain-specific generic actions and data structures in PROTÉGÉ. | For each interpreter a set of generic actions and data structures is predefined. They correspond to some of the roles of the method. |
| Instantiated actions and their concrete data | Domain specific knowledge filled into a predefined data structure is used by the interpreter to instantiate generic actions at run-time. | Defining a domain layer and linking it to an interpretation model yields a conceptual model. | Domain specific generic actions are instantiated in p-OPAL. | Domain specific actions can be defined according to the schema defined by the generic actions. At the same time data is filled in. |
| Selection criteria for generic actions | Fixed. | Domain independent, on the task layer. | Defined in PROTÉGÉ. | Fixed. |
| Selection criteria for instantiated actions | Domain specific knowledge. | None, all control happens on the task layer. | Defined in p-OPAL. | Domain specific knowledge. |

In table 2, "predefined" entities belong to the generic part, while "defined" ones may be supplied by the knowledge engineer or the expert. In the MT approach, a very clear distinction is drawn between defining and instantiating. To elucidate it, we included the respective editors in the table.

The following table explicates for all approaches which types of knowledge are generic and which must be supplied by the knowledge engineer or the expert.

Table 3 confirms hypothesis (II) of the introduction: the problem solving method (as specified in the *generic parts* ) is predefined, i.e. it is specified in a problem specific, but task-independent way. And the parts to be filled in are always task-specific. However, the borderline between the generic and the task-specific parts varies. Except for the MT approach, the generic parts always include the selection criteria, generic actions and data structures. Complementary, in all approaches the instantiated actions and the concrete data must be entered. These remarks apply to the KADS approach only if predefined interpretation models are selected. Alternatively, KADS allows to construct models from scratch using the four layer modelling language. In this case, the modelling language is the only generic part, and the problem solving method (the interpretation model) must be specified completely by the knowledge engineer. This brings us to the more fundamental differences of the four approaches.

Table 3: Origin of the different types of knowledge in the four approaches.

| Generic vs. task specific | GT approach | KADS | MT approach | RLM approach |
|---|---|---|---|---|
| Generic part | Problem solving method, generic actions and data structures, selection criteria for generic actions. | Problem solving method. | Problem solving method. | Problem solving method, generic actions and data structures, selection criteria for generic actions. |
| Task specific knowledge filled in by KE | Data and selection criteria to instantiate actions. | Instantiated actions and data. | Generic actions and data structures, selection criteria for generic actions. | Selection criteria for instantiated actions and data. |
| Task specific knowledge filled in by experts | | | Instantiated actions, selection criteria for instantiated actions. | Instantiated actions and data, selection criteria for instantiated actions. |

We already deplored that the models are not always as explicit as desirable. In fact, only KADS provides a (semi-formal) modelling language for describing arbitrary models, and this language was actually used to specify all the interpretation models mentioned in the previous section. The other three approaches all provide a set of isolated, hand-coded models. Each of the models is described either using a particular modelling vocabulary, or in prose that is not reflected in the implementation.
While KADS is the only approach providing a general modelling language, it is also the only one whose models are not operational. These are two sides of the same coin. The generality of the language makes it not directly implementable. As shown in figure 1 the gap between the modelling language and the implementation must be bridged by design models.

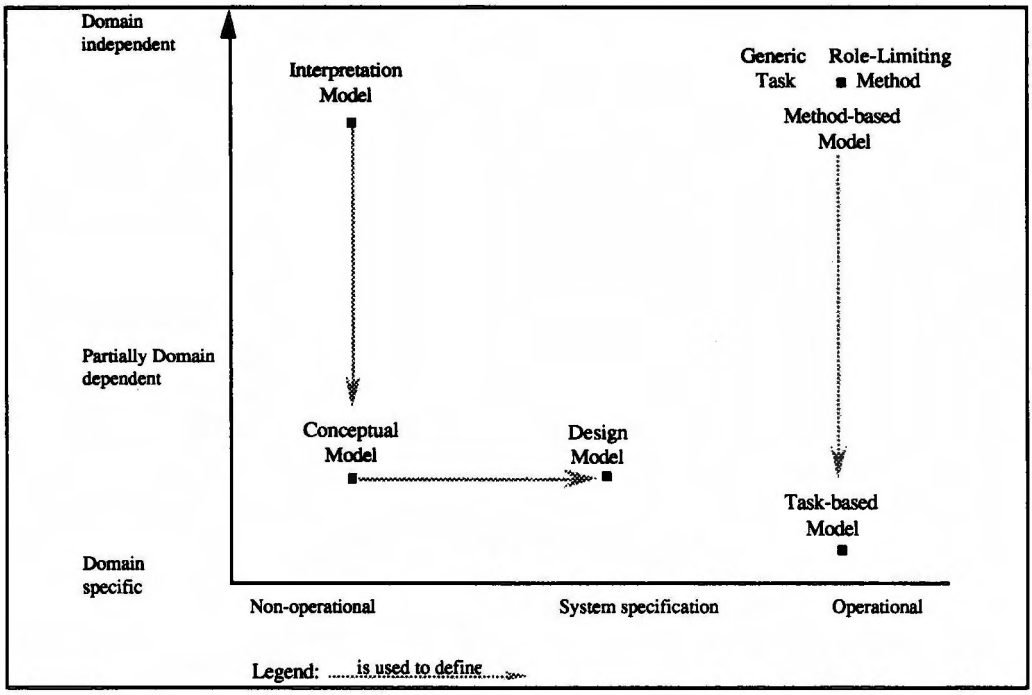



Figure 1: Different problem solving models ordered according to their level of specialization and their operationality.

While the KADS conceptual model is supposed to model the expert rather than the system, the design model describes the system, though no longer on the modelling level. None of the other approaches directly models the expert, as they provide a fixed set of modelling level descriptions of operational systems. The problem-solving model of the operational system may coincide with the expert's behavior.

If operational systems are written in the same implementation language, they can be combined, even if a combination is not supported at the modelling level. E.g. generic tasks can communicate via message passing. This is different in KADS: Since there is a general modelling language and no implementation to be considered, their models are arbitrarily combinable.

Another crucial aspect of a modelling approach is its support for choosing the right model. All approaches describe the knowledge needs resp. input in prose. This corresponds to the task-specific knowledge in the latter two rows of table 3. Only the GT approach also specifies the output, and KADS offers the inference structures, i.e. a control-independent description of the elementary actions and their data structures. This seems to be a rather sad affair. A taxonomy of the various models with decision criteria for each branch would be a great help. Attempts in this direction are KADS' library of interpretation models or the distinction between class 1, 2 and 3 design in the GT approach.

The differences described so far are summarized in table 4. The entries of the last row heavily depend on the knowledge acquisition tools supporting the approach. Only in the MT approach editors can be used to refine the model. In the GT and KADS approaches this job is entirely left to the knowledge engineer, but if suitable tools were built, as was done in the RLM approach, the expert could easily fill in instantiated actions or data. So all approaches support the first aspect of hypothesis (III). Knowledge acquisition is guided by the knowledge needs the systems have. They can be further specialized in task specific terms, like in p-OPAL, to close the terminology gap between expert and system. Knowledge acquisition interfaces, e.g. in the RLM approach, point to missing knowledge. However, the different approaches do not necessarily lead to better structured expert systems. In cases where the acquired knowledge is implemented in a general purpose language that does not reflect the modeling constructs modifications and maintenance will be almost as difficult as in conventional approaches.

Table 4: Major differences between the four approaches.

| | GT approach | KADS | MT approach | RLM approach |
|---|---|---|---|---|
| Method definition language | No | Yes | No | No |
| Operational | Yes | No | Yes | Yes |
| Model of ... | System | Expert or system | System | System |
| Combinable | Via message passing | Yes | No | No |
| Model characterization | Verbal description, input/output. | Inference structure, knowledge needs, verbal descriptions. | Verbal description | Verbal description, knowledge needs. |
| Intended users | KE/expert | KE/expert | KE(PROTÉGÉ) and expert (p-OPAL). | Expert |

## 5. Summary

Our comparison has shown that there are major differences between the modeling approaches. It is obvious that KADS is far more flexible than other approaches, which is due to its total lack of operational semantics. The other three approaches differ mainly in their ways to represent the models in their knowledge base (GT and MT do have an internal representation, RLM does not), and in their capability to reduce the terminological gap between the expert's vocabulary and the system's needs (MT uses domain terminology, the others don't).

In the future, high flexibility is needed so that the knowledge engineer can compose, modify or design new models at need. Internal representations that explicate the problem solving models are preconditions for problem solvers with reflective capabilities that go beyond the symbol level type of reasoning. It is obvious that the use of task-specific vocabulary is a big step towards clients' maintenance of the knowledge base.

A combination of the four approaches looks very promising: the flexibility of KADS to define models, the explicit representation of the operational modeling constructs as it is done in the GT approach, the role-driven knowledge acquisition capabilities of the RLM approach, combined with editors that separate the task of the knowledge engineer from the responsibility of the domain expert, as it is done in the MT approach.

## Acknowledgements

We thank Georg Klinker, Mark Musen and Hans Voß for their very helpful comments on earlier versions of this paper.

## References

1. Allen, James F. Maintaining Knowledge About Temporal Intervals. *Communications of the ACM 26*, 11 (1983), 823 - 843.

2. Brachman, Ronald J. On the Epistemological Status of Semantic Networks. In *Associative Networks: Representation of and use of knowledge by computers.* Academic Press, Findler, N.V., New York, 1979.

3. Breuker, Joost, Wielinga, Bob, van Someren, Maarten, de Hoog, Robert, Schreiber, Guus, de Greef, Paul, Bredeweg, Bert, Wielemaker, Jan, Billaut, Jean-Paul, Davoodi, Massoud and Hayward, Simon Model-Driven Knowledge Acquisition: Interpretation Models. Tech. Rept. Esprit Project 1098, Deliverable Task A1, University of Amsterdam, Social Sciences Informatics, Amsterdam, 1987.

4. Breuker, Joost and Wielinga, Bob Models of Expertise in Knowledge Acquisition. In *Topics in Expert System Design, Methodologies abd Tools.* North-Holland, Guida, Giovanni and Tasso, Carlo, pp. 265 - 295, Amsterdam, 1989.

5. Brown, David C. and Chandrasekaran, B. *Design Problem Solving: Knowledge Structures and Control Strategies,* Pitman, London , Research Notes in Artificial Intelligence(1989).

6. Bylander, Tom and Chandrasekaran, B. Generic Tasks for Knowledge-Based Reasoning: The 'right' Level of Abstraction for Knowledge Acquisition. Tech. Rept. 87-TB-KNOWAC, Technical Research Report, Ohio State University , Laboratory for Artificial Intelligence Research, Columbus, 1987.

7. Chandrasekaran, B. Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design. *IEEE Expert* (Fall 1986), 32 - 30.

8. Chandrasekaran, B. Generic Tasks as Building Blocks for Knowledge-Based Systems: the diagnosis and routine design examples. *The Knowledge Engineering Review* (Octobre 1988), 183 - 210.

9. Clancey, William Heuristic Classification. *Artificial Intelligence 27*(1985), 289 - 350.

10. Eshelman, Larry MOLE: A Knowledge-Acquisition Tool for Cover-and-Differentiate Systems. In *Automating Knowledge Acquisition for Expert Systems.* Kluwer Academic Publishers, Marcus, Sandra, Ch. 3, pp. 37 - 80, Boston, 1988.

11. Forbus, Kenneth D. Qualitative Process Theory. *Artificial Intelligence* (1985), 85 - 168.

12. Karbach, Werner, Voß, Angi and Tong, Xuejun Filling in the Knowledge Acquisition Gap: via KADS' Models of Expertise to ZDEST-2's Expert Systems. In *Proceedings of EKAW88*, Boose, John H., Gaines, Brian and Linster, Marc, GMD, St. Augustin, 1988, pp. 31/1 - 31/17.

13. Klinker, George, Boyd, Casey, Dong, David, Maiman, Jon, McDermott, Jofn and Schnellbach, Robert Building Expert Systems with KNACK. *Knowledge Acquisition 1*, 3 (September 1989), 299 - 320.

14. Kodratoff, Yves and Tecuci, Gheorghe Learning at Different Levels of Knowledge. In *Proceedings of EKAW88*, Boose, John H., Gaines, Brian R. and Linster, Marc, GMD, St. Augustin, 1988, pp. 3/1 - 3/17.

15. Kuipers, Benjamin Commonsense Reasoning about Causality: Deriving Behavior from Structure. *AI Journal* (1984), 169 - 203.

16. Marcus, Sandra A Knowledge-Acquisition Tool for Propose-and-Revise Systems. In *Automating Knowledge Acquisition for Expert Systems*. Kluwer Academic Publishers, Marcus, Sandra, Ch. 4, pp. 81 - 124, Boston, 1988.

17. Marcus, Sandra Understanding Decision Ordering from a Piecemeal Collection of Knowledge. *Knowledge Acquisition 1*, 3 (Septembre 1989), 279 - 298.

18. McDermott, John Preliminary Steps Toward a Taxonomy of Problem-Solving Methods. In *Automating Knowledge Acquisition for Expert Systems*. Kluwer Academic, Marcus, Sandra, Ch. 8, pp. 225 - 256, Boston, 1988.

19. Musen, Mark A., Fagan, L. M., Combs, D. M. and Shortliffe, E. H. Use of a domain-model to drive an interactive knowledge-editing tool. *International Journal of Man-Machine Studies 26*(1987), 105 - 121.

20. Musen, Mark A. Conceptual Models of Interactive Knowledge Acquisition Tools. In *Proceedings of EKAW88*, Boose, John H., Gaines, Brian R. and Linster, Marc, GMD, St. Augustin, 1988, pp. 26/1 - 26/15.

21. Musen, Mark A. *Automated Generation of Model-Based Knowledge Acquisition Tools*, Pitman Publishing, London , Research Notes in Artificial Intelligence(1989).

22. Norman, D. Some Observations on Mental Models. In *Mental Models*. New Yersey: Hillsdale, Gentner, D., 1983.

23. Schreiber, Guus, Bredeweg, Bert, Davoodi, Massoud and Wielinga, Bob Towards a Design Methodology for KBS. Tech. Rept. 97, ESPRIT Project 1098, Deliverable D8, VF Memo , University of Amsterdam, Amsterdam, 1987.

24. Schreiber, Guus, Breuker, Joost, Bredeweg, Bert and Wielinga, Bob Modelling in KBS Developement. In *Proceedings of EKAW88*, Boose, John H., Gaines, Brian R. and Linster, Marc, GMD, St. Augustin, 1988, pp. 7/1 - 7/15.

25. Voß, Hans and Linster, Marc Interval-Based Envisioning in HIQUAL. *Applied Artificial Intelligence, Special Issue on Causal Modeling 2-3*(1989), 17 - 43, Also published in: W. Horn(ed.):Causal AI Models - Steps Toward Applications, Hemisphere Publ., New York, 1990, p. 17 - 43.

# Comparison of Inductive and Naive Bayesian Learning Approaches to Automatic Knowledge Acquisition

**Igor Kononenko**
**University of Ljubljana**
**Faculty of Electrical and Computer Engineering**
**Tržaška 25, 61001 Ljubljana, Yugoslavia**

**Abstract.** The system for inductive learning of decision trees Assistant and naive Bayesian classifier were used as automatic knowledge acquisition tools in order to develop expert systems in four medical diagnostic problems. The accuracy of automatically acquired knowledge from stored data records is compared and the interpretation of the knowledge and the explanation ability of the classification process of each system is discussed. Surprisingly, the naive Bayesian classifier is superior to Assistant in classification accuracy and explanation ability, while the interpretation of the acquired knowledge seems to be equally valuable.

**Keywords:** machine learning, induction of decision trees, naive Bayesian classifier, explanation, medical diagnostics

## 1 Introduction

Inductive learning is well known approach to automatic knowledge acquisition for expert systems which overcomes the so called Feigenbaum bottleneck of knowledge acquisition from human experts [10,15]. Instead of time consuming process of extracting the knowledge from experts and professional literature an inductive learning system can be used to generate the knowledge, usually in the form of decision rules, from the concrete problems solved by some experts in the past.

In this paper two approaches to automatic knowledge acquisition are compared: ID3-like inductive learning system Assistant [1,2] and naive Bayesian classifier. Basically, the two approaches are much different, according to knowledge representation as well as the explanation ability. From training examples Assistant generates a decision tree while knowledge generated by naive Bayesian classifier is simply a table of conditional and prior probabilities.

A new problem can be solved using a decision tree by following the appropriate path from the root to the leaf of a decision tree. The path corresponds to the decision rule and itself represents the explanation of the solution of the problem. On the other hand, the naive Bayesian classifier solves the new problem by appropriately combining the probabilities from the table. The explanation of the solution is the sum of information gains from different attributes (features, parameters, descriptors) to the conclusion.

In the next section both methodologies are briefly described. In section 3 the experiments in four medical diagnostic problems are described and in section 4 the different knowledge representations are discussed as well as the estimation of both approaches by human experts. Finally in section 5 we give some conclusions and propose further research.

## 2 Assistant and Naive Bayesian Classifier

The classification problem discussed in this paper is the following: given a set of training instances, each described with a set of attributes and each belonging to exactly one of a certain number of possible classes, learn to classify new, unseen objects. In addition, each attribute has a fixed number of possible values.

Let $C$ represent one of the possible classes. Let $V_{iJ_i}$ be a Boolean variable having value 1 if the current instance has value $J_i$ of i-th attribute and 0 otherwise. Then the conditional probability of class $C$ given the values of all attributes is given with the following formula derived from the Bayesian rule [4] (for brevity conditions $V_{ij} = 1$ will be written simply as $V_{ij}$):

$$P(C|V_{1,J_1},\ldots,V_{n,J_n}) = P(C)\prod_{i=1}^{n} Q_i(C,J_i) \tag{1}$$

where

$$Q_i(C,J_i) = \frac{P(V_{i,J_i}|C,V_{1,J_1},\ldots,V_{i-1,J_{i-1}})}{P(V_{i,J_i}|V_{1,J_1},\ldots,V_{i-1,J_{i-1}})} = \frac{P(C|V_{1,J_1},\ldots,V_{i,J_i})}{P(C|V_{1,J_1},\ldots,V_{i-1,J_{i-1}})} \tag{2}$$

and $P(C)$ is the prior probability of class $C$. It was shown in [6] that the classification with ID3 like [13] inductive learning system can be described with (1). The basic learning algorithm is as follows:

```
P'(C) := P(C);
if (P'(C) = 1) or (P'(C) = 0) or no more attributes
   then generate a leaf
   else
    begin
      select attribute i that minimizes the expected entropy;
      split the learning instances according to attribute's values;
      for each attribute's value J_i do
         recursively construct a subtree with
           P'(C) := P'(C) * Q_i(C, J_i)
   end
```

Assistant [1,2] was derived from ID3 and incorporates mechanisms for dealing with incomplete and noisy domains. The main advantages with respect to ID3 are on-line binarization

of attributes and pre and post-pruning techniques for discovering and pruning the unreliable parts of decision trees.

From (1) the naive Bayesian classifier, as used by Bratko and Kononenko [1], is obtained if the independence of attributes is assumed. (1) remains unchanged except that factors $Q_i$ defined with (2) are replaced with $Q_i'$ (we will refer to changed equation (1) with (1')):

$$Q_i'(C, J_i) = \frac{P(V_{i,J_i}|C)}{P(V_{i,J_i})} = \frac{P(C|V_{i,J_i})}{P(C)} \tag{3}$$

The probabilities necessary to calculate (3) are approximated with relative frequencies from the training set. A new object is classified by calculating the probability for each class using equation (1') and the object is classified into the class that maximizes the calculated probability.

Cestnik [3] has shown that the kind of approximation of probabilities in (3) considerably influences the classification accuracy of naive Bayesian classifier. Let $N(C, V_{iJ_i})$ be the number of training instances with $J_i$-th value of i-th attribute and belonging to class $C$ and $N(V_{iJ_i})$ the number of training instances with $J_i$-th value of i-th attribute. Usually the probability is approximated with relative frequency, i.e.

$$P(C|V_{i,J_i}) = \frac{N(C, V_{i,J_i})}{N(V_{i,J_i})} \tag{4}$$

However, if the training set is relatively small, the corrections are needed with respect to the assumption of initial distribution [5]. Cestnik [3] used the following formula stemming from the assumption, that initial distribution of classes is equal to $P(C)$:

$$P(C|V_{i,J_i}) = \frac{N(C, V_{i,J_i}) + 2P(C)}{N(V_{i,J_i}) + 2} \tag{5}$$

where the probability of class C is calculated using the Laplace law of succession [4]:

$$P(C) = \frac{N(C) + 1}{N + 2} \tag{6}$$

Cestnik has shown some nice properties of using approximation (5) in formula (3) and has shown experimentally, that naive Bayesian classifier using approximation (5) performs significantly better than if (4) is used.

# 3 Experiments in Medical Diagnostics

We experimented with two learning systems described in the previous section in four medical diagnostic problems: diagnosing the location of primary tumor, prognosing the recurrence of breast cancer, diagnosing thyroid diseases and rheumatology. The characteristics of data sets used in our experiments are summarized in table 1. The data was collected at the University Medical Center in Ljubljana.

One run was performed by randomly selecting 70% of instances for learning and 30% for testing. Results are averages of 10 runs and are given in table 2. The average percent of

*Table 1:* Characteristics of four medical domains.

| domain name | # attributes | # diagnoses | # cases | entropy | maj. class |
|---|---|---|---|---|---|
| primary tumor | 17 | 22 | 339 | 3.64bits | 25% |
| breast cancer | 10 | 2 | 288 | 0.72bits | 80% |
| thyroid diseases | 15 | 4 | 884 | 1.59bits | 56% |
| rheumatology | 32 | 6 | 355 | 1.70bits | 66% |

*Table 2:* The comparison of performance of different classifiers in four medical domains.

| classifier | primary tumor | breast cancer | thyroid | rheumatology |
|---|---|---|---|---|
| naive Bayes | 49% 1.59bits | 78% 0.15bits | 73% 0.85bits | 64% 0.46bits |
| Assistant | 44% 1.38bits | 77% 0.07bits | 73% 0.87bits | 61% 0.46bits |
| physicians | 42% 1.22bits | 64% 0.05bits | 64% 0.59bits | 56% 0.26bits |

correct guesses is given and the average information content [9] per answer is added to the table. Information content is the measure that eliminates the influence of prior probabilities of classes and can be applied to various kinds of incomplete and probabilistic answers. For completeness, its definition is provided in the appendix. This measure is necessary as in each domain a classifier that classified each instance into the majority class (see table 1) would achieve high classification accuracy.

The results are compared to the accuracy of physicians experts which. The performances of physicians are the averages of four physicians experts in each domain that were tested in University Medical Center in Ljubljana. Both Assistant and naive Bayesian classifier outperformed physicians specialists but naive Bayes was also better than Assistant.

# 4 Knowledge Representation and Explanation Ability

The knowledge generated by Assistant is in the form of a decision tree. The root of a tree is usually the most important attribute for the given classification problem. In fact it is the attribute which minimizes the expected entropy, i.e. the expected amount of information, necessary to classify a new object. The physicians typically agree that the attribute at the root of the tree contains the most important information. Table 3 represents the ordering of attributes in the primary tumor domain by Assistant and by a physician specialist with respect to their significance.

The physicians felt that the structure of the top of decision trees makes sense and is an interesting representation of the decision process. However, although a decision tree outperformed the physicians experts with respect to diagnostic accuracy the physicians were not prepared to use them in practice. The rules (paths from the root to the leaves of the tree)

*Table 9*: The comparison of ordering of attributes with respect to their significance in the primary tumor domain by Assistant and by human expert.

| physician's ordering | Assistant's ordering |
|---|---|
| histologic type | histologic type |
| sex | sex |
| brain | axillar |
| lung | bones |
| neck | differentiation |
| bones | diaphragm |
| age | peritoneum |
| skin | neck |
| peritoneum | lung |
| ... | ... |

were too short, containing only few, although most informative, attributes and were too purely describing the patient to make the reliable decision [11]. The physicians typically use all available information to make a decision and they are also able to estimate the reliability of the diagnosis. If the reliability is not high enough then additional examinations are needed.

On the other hand, the knowledge generated by naive Bayesian classifier is simply a table of prior and conditional probabilities approximated with relative frequencies from the training set. Physicians evaluated the table as useful information.

Besides, the classification process by naive Bayesian classifier can be naturally interpreted using the definition of information [14]. The minus logarithm of (3) is interpreted as the information gain by i-the attribute to the conclusion that an object belongs to class $C$ [7]. The minus logarithm of (1') gives:

$$-log_2 P(C|V_{1,J_1}, \ldots, V_{n,J_n}) = -log_2 P(C) - \sum_{i=1}^{n} log_2 Q_i(C, J_i) \qquad (7)$$

which is interpreted as the amount of information necessary to find out that an object belongs to class $C$ given the values of all attributes. That amount of information before the classification is equal to $-log_2 P(C)$. Therefore the right hand side of (7) is interpreted as the sum of information gains from all attributes to the conclusion that an object belongs to class $C$.

Physicians found such explanation of classification as natural and similar to their classification. They also sum up the evidence for/against a diagnosis. They were immediately prepared to use the naive Bayesian classifier with the incorporated facility of explaining the classification by means of equation (7) as a tool for enhancing the reliability of a diagnostic process.

# 5 Discussion

When classifying new objects with a decision tree the values of only few attributes are examined, those corresponding to the path from the root of a tree to the leaf. In some cases this is an advantage as the values of attributes can be expensive to obtain. However, if attributes are available it is better to take them into account. Besides, if the value for some attribute in the tree is missing (e.g. for the attribute at the root of the tree) the classification becomes unreliable [1]. Physicians felt that the patient cannot be reliably diagnosed with only few attributes [11].

On the other hand, naive Bayesian classifier takes into account all the available attributes. If the value of an attribute is missing, such attribute is simply omitted from equation (1'). The problem with naive Bayes is the independence assumption. In some cases this may be too unrealistic assumption. But it seems that in the data used by human experts there are no strong dependencies between attributes because attributes are properly defined. With the independence assumption the reliability of approximating factors (3) with relative frequencies is much greater than the reliability of approximating factors (2).

This is supported with experimental results. Naive Bayesian classifiers despite its naiveness achieved better classification accuracy. There is a trade-off between the reliability of approximating probabilities and the errors due to the independence assumption. Further work is in progress to develop semi-naive Bayesian classifier, that will try to optimize this trade-off.

Assistant derives from training instances explicit if-then rules tied together in a decision tree. Rules are easy to understand and can be used without a computer, simply listed on the paper. Assistant tends to generate few general rules. A decision tree represents the global structure of the classification problem.

The naive Bayesian classifier's knowledge can be viewed as implicit, as it can not be directly applied without calculation. A table of (conditional) probabilities represents a detailed and distributed knowledge without any global picture about the structure of the problem. Naive Bayes performs like if it knew the rules. However, the rules that it uses for classifying objects are typically different for each object from a domain. Therefore, naive Bayes uses a lot of specialized rules.

Such kind of knowledge is typical for neural networks [12]. In it was shown in [8] that the naive Bayesian classifier can be naturally implemented with the Bayesian neural network. Such a network is able to explain its decisions while preserving the advantages of neural networks: multidirectional classification, robustness with respect to noisy and missing data and fast and incremental learning. Induction of decision trees is relatively slow, non-incremental and sensitive to missing and noisy data.

Better classification accuracy is not enough for an expert system to be used in practice. It must be able to explain its decisions and it must be flexible enough to provide all possible alternatives. The overall physician's impression was that the form of naive Bayesian classifier's explanation typically replicates their way of diagnosing, i.e. the summation of evidence for/against the diagnosis. In the 'breast cancer' domain physicians found certain decisions as a new information interesting for further investigation. Currently the developing of an expert system in primary tumor and breast cancer is in progress.

## Acknowledgements

Physicians specialists Sergej Hojker, Vlado Pirnat and Matjaz Zwitter from University Medical Center in Ljubljana estimated the correctness of explanations and their understandability. They also provided the data used in our experiments. This research was done in Artificial Intelligence Laboratory at the Faculty of Electrical & Computer Engineering in Ljubljana and was supported by Slovenian Research Community. I would like to thank Prof. Ivan Bratko for providing the environment for efficient scientific work.

## References

[1] I.Bratko & I.Kononenko, Learning Rules from Incomplete and Noisy Data, in B. Phelps (ed.) *Interactions in Artificial Intelligence and Statistical Methods*, Hampshire: Technical Press, 1987.

[2] Cestnik B., Kononenko I.& Bratko I., ASSISTANT 86 : A knowledge elicitation tool for sophisticated users, in: I.Bratko, N.Lavrac (eds.): *Progress in Machine learning*, Wilmslow: Sigma Press, 1987.

[3] Cestnik B., Estimating Probabilities: A Crucial Task in Machine Learning, *Proc. European Conf. on Artificial Intelligence*, Stockholm, August 1990 (in press).

[4] Good, I.J., *Probability and the Weighing of Evidence*, London: Charles Griffin, 1950.

[5] Good I.J., *The Estimation of Probabilities*, Cambridge: M.I.T. Press, 1965.

[6] Kononenko I., ID3, Sequential Bayes, Naive Bayes and Bayesian Neural Networks. *Proc. 4th European Working Session on Learning*, Montpeiller, 4-6 December 1989, pp.91-98.

[7] Kononenko I., Interpretation of neural networks decisions, *Proc. IASTED Intern. Conf. Expert Systems & Applications*, Zurich, 26-29 June 1989, pp. 224-227.

[8] Kononenko I., Bayesian neural networks, *Biological Cybernetics*, Vol. 61, pp.361-370, 1989.

[9] Kononenko, I. & Bratko, I., Informativity Based Evaluation Criterion for Classifier's Performance. *Machine Learning Journal.* (submitted). (also: *ISSEK Workshop*, Udine:Italy, Sept. 1989).

[10] Michie D., Machine learning and knowledge acquisition. In: *International Handbook of Information Technology and Automated Office Systems.* North-Holland, Elsevier Science Publishers, 1985.

[11] Pirnat V., Kononenko I., Janc T. & Bratko I., Medical Estimation of Automatically Induced Decision Rules, *Proc. of 2nd Europ. Conf. on Artificial Intelligence in Medicine*, City University, London, August 29-31 1989, pp.24-36.

[12] Quinlan, J.R., Discovering rules by induction from large collections of examples, in D.Michie (ed.): *Expert systems in the Micro Electronic Age*, Edinburgh University Press, 1979.

[13] Rumelhart, D.E. & McClelland, J.L. (eds.), *Parallel Distributed Processing, Vol. 1: Foundations.* Cambridge: MIT Press, 1986.

[14] Shannon, C.E. & Weaver, W., *The mathematical theory of communications.* The University of Illinois Press, Urbana, 1949.

[15] Steels L., Second generation expert systems. *Future Generation Computer Systems*, Vol.1, pp.213-221, 1985.

# Appendix: Definition of the Information Content of an Answer

A fair evaluation criterion has to exclude the influence of the prior probabilities of classes which may enable a completely uninformed classifier to trivially achieve high classification accuracy. The measure of information content of the classifier's answer defined below excludes the influence of prior probabilities, deals with various types of imperfect or probabilistic answers and can be used also for comparing the performance in different domains. Its interpretation is natural.

**Definition:**

Let the correct class of an instance be $C$, $P(C)$ be the prior probability of class $C$ and $P'(C)$ the probability of class $C$ returned by a classifier. The information content $I$ of classifier's answer is defined as follows (note that in our experiments only exact classification is considered, i.e. $P'(C)$ is always either 1 or 0):

a) if $P'(C) > P(C)$ then

$$I = -log_2 P(C) + log_2 P'(C) \quad [bits]$$

i.e., the amount of obtained information is the entire amount of information necessary to correctly classify an instance into class $C$ minus the remainder of information necessary to correctly classify that instance.

b) if $P'(C) = P(C)$ then $I = 0[bits]$

i.e., the system didn't change the prior probability of the correct class therefore we did not obtain any information.

c) if $P'(C) < P(C)$ then

$$I = -(-log_2(1 - P(C)) + log_2(1 - P'(C))) \quad [bits]$$

i.e., the amount of information returned by the system is the entire amount of information necessary to decide that an instance does not belong to class $C$ minus the remainder of information necessary to make that decision. As this information is in fact wrong the information content of the system's answer in this case is defined as negative.

# On the use of a formalized generic task model in knowledge acquisition

Wojciech Kowalczyk
Jan Treur

*Vrije Universiteit*
*Department of Mathematics and Computer Science*
*Artificial Intelligence Section*
*de Boelelaan 1081*
*1081 HV Amsterdam*
*The Netherlands*

**Abstract** The role of a formalized generic task model may be viewed in the light of the development to automated support to knowledge engineering. If it is acknowledged that model-driven knowledge acquisition is a useful method in knowledge engineering, and that automated support for it is desirable, then a formalized notion of a generic task model is indispensable. The research as reported in this paper gives an answer on how such a formalization can be made, resulting in a notion of a generic task model that is both syntactically and semantically clear and well-defined. This notion is related to the framework for DEsign and Specification of Interacting REasoning modules (DESIRE) that has been developed in our research group. Moreover some of the possible applications to knowledge acquisition are sketched, and some of our experiences are discussed.

## 1 Introduction

The development process of a (modular) knowledge-based system may roughly be described by a number of activities and resulting products (documents, implementations) such as:

- describing the expert knowledge that is involved (by means of a conceptual model),
- design and (formal) specification of a task model (anticipating the global system architecture),
- refining the task model by filling in detailed domain knowledge in its different components,
- design and (formal) detailed specification of the system,
- implementation of a (prototype) system, using an AI-tool such as an Expert System Shell, a suitable AI-language, or any other programming language.

Here every activity except the last one is independent of the chosen tools for implementation, and may involve (additional) knowledge acquisition.

The order in which these activities are done may vary. One may carry out these activities in some order, for example in the given order, or in the order that starts with establishing a task model and then using this model as a driving force in the early stages of the knowledge acquisition (model-driven knowledge acquisition; for instance see [3]). In practice, however, often the activities are not carried out sequentially, but rather iterative, by providing progressive refinements during the process of iteration. In particular, not seldom endusers find it difficult to express the functional

requirements they would pose, since they have no imagination of the possibilities they may choose. Especially an iterative development process may help in making these functional requirements explicit step by step, and this may even involve offering (partial) preliminary prototype implementations of the system.

An important disadvantage of an iterative development process may be that the prototype implementations can have a very conservative impact on the design of the final system. Such a conservative impact will have some relation (proportional) to the effort needed to build a (new) prototype or a system. This disadvantage may be minimized by use of the following two principles:

- concerning the system architecture the stress is not put on the implementation, but on the design and specification of the system; creating specifications and modifying them should be supported by a library of already existing (parts of) specifications and by supporting software tools,

- similarly, implementing prototypes from a specification should be supported by software tools and a library of reusable modules and task-specific shells.

In the approach of our research group both principles have been taken as a point of departure in the research projects of recent years (in particular in the Dutch projects SIX, SKBS-A3, DISKUS and IIICAD, and in the ESPRIT-project ROCOCO). Until now this resulted in the specification framework for modular knowledge-based systems called DESIRE (framework for DEsign and Specification of Interacting REasoning modules, described in section 2) and in our concept of a formalized generic task model (section 3). Moreover first versions of the following software tools have been developed and are now being tested extensively:

1. to support making DESIRE-specifications of a modular knowledge-based system:
   - *syntax checker*
   - checkers on *correctness* and *completeness* of a specification

2. to support making implementations out of DESIRE-specifications:
   - an *implementation environment* for modular knowledge-based systems consisting of
     . an environment to implement separate reasoning modules
     . a general manager component for module handling
     . task-specific shells related to given generic task models as established in a number of domains
   - a *compiler* to translate specifications to code in this implementation environment

The above mentioned tools have been implemented in a UNIX environment, on a SUN SPARC station. The implementation environment has been built as an extension (in C) of the popular Expert System Development Environment NEXPERT OBJECT.

Our first experiences with our approach suggest that it is possible to generate new modified prototype implementations from adapted DESIRE-specifications within a very short time. As far as creating specifications is concerned, the advantages of reusing module specifications and (parts of) generic task models has shown to be effective, especially for tasks that are relevant in the domains of the above mentioned projects, namely classification tasks and design, planning, and scheduling tasks.

In summary, by using adequate supporting tools the main disadvantages of an iterative development process may be minimized, thus opening the possibility of using a systematic methodology, where (partial) specifications and implementations of generic task models and prototype systems may play a more solid role in knowledge acquisition.

## 2 Modular knowledge-based systems and formal specification

A modular knowledge-based system consists of a number of modules. Among these there may be *conventional modules*; the other, knowledge-based modules are called *reasoning modules*. Each module can operate on a set of related data, called here an *information state*. As a result of an activation of a module, the information state is modified (extended by new, derived information) providing some data which may be relevant for other modules. Because we allow the format of the information states of different modules to be different, some special functions, called *transformations*, are used to modify the format of the information provided by one (or more) module(s) in such a way that this information can be used as an input for another module. This enables one to reuse modules from specifications or implementations of other systems without having to change their language.

The overall dynamics of the system is controlled by a component called a *supervisor*. Its role is to activate modules in a proper order (which may depend on a specific state of the whole system) and in a proper way (to activate a module an inference strategy should be provided first). Moreover the supervisor is responsible for the execution of appropriate transformations. A specification of the behaviour of the whole system (in the form of some *supervisor rules* which express knowledge about which modules should be activated in which order) is called a *control structure*, and can be viewed as a kind of knowledge base which is provided to the *general manager* which is able to use this knowledge and may carry out actions accordingly. In other words, the general manager is a generic component which is able to interpret the control structure of the system; when instantiated with specific knowledge it becomes the supervisor of the system.

Now we will discuss the above concepts in more detail. Except of giving a detailed description of these concepts, our specification framework DESIRE will be outlined; the issue of *what* should be formally specified, and *how* can it be specified is central for our considerations. For a discussion of some issues outlining a background of the framework the reader is referred to [24, 25, 26, 29]; also an introduction to logic, based for example on [7], may be useful.

### 2.1 The concept of a module

A reasoning module is the basic component of a modular knowledge-based system. It is supposed to contain (domain) specific knowledge which is relevant to a given type of primitive task. *Primitive* means here, that a uniform inference strategy applied to the module's knowledge base should be sufficient to solve the task. We may think of a module as an autonomous knowledge-based system. It contains a *knowledge base* in the form of a set of rules, which describe how atomic statements about the world situation of concern are related to each other. To use this knowledge, one has to provide to the module a set of input facts (in the form of atoms with their truth values) and to specify a task for an inference mechanism, e.g., by giving a set of atoms which have to be derived (the target set) or a set of atoms which are used as data for data-driven inference (the data set). As a result some new facts can be derived, extending the amount of information about the current situation. This implies that two types of information play a role in the process carried out by a module: *object-level information* (about the world situation that is concerned) and *meta-level information* (about the state of the process of the module itself). In the two sections below these two types are treated subsequently.

#### 2.1.1 Object-level aspects of a module

The object-level of a module can be viewed as a device which modifies information states; the *signature* provides an alphabet for the language used in the module, atoms provide a set of atomic pieces of information, the information state determines the current status of these atoms (e.g., **true**, **false**, or **unknown**), and the knowledge base describes logical interrelations between the atoms.

Moreover, the signature can be split into three subsignatures which specify the format of *input*, *output*, and *internal* facts. The role of input and output signatures is obvious; the internal signature specifies the format of facts, which can be used inside the module and are not relevant for any other module. For example, a module supposed to reason about some illnesses may use an input parameter **t** to denote the temperature of a patient, an internal atom **fever** to denote the "abstracted" temperature of the patient, and an output atom **influenza**, to denote the possible outcome of the diagnosis. The rules $t > 38 \rightarrow$ **fever**, and **fever** $\wedge \ldots \rightarrow$ **influenza** describe then a piece of the relevant knowledge. Activation of a module (by applying an inference mechanism) results in a new information state. Formally, a DESIRE-specification of a module should contain:

- a *name* of the module;

  this name can be used in the specification of the control structure of the whole system,

- an *input signature* of the module;

  this signature provides the format of input atoms,

- an *output signature* of the module;

  this signature provides the format of output atoms,

- an *internal signature* of the module;

  this signature provides the format of internal atoms.

(Note that input and output signatures may overlap. The three signatures determine a global signature of the module. The knowledge base is expressed in terms of this module signature.)

- *knowledge base*;

  a list of rules which express logical relations between atoms.

The knowledge base is specified by a list of formulas written in the logical language based on the given signature of the module. In DESIRE we specify knowledge in the following *rule-format*:

$$a_1 \wedge \ldots \wedge a_m \rightarrow b_1 \wedge \ldots \wedge b_n,$$

where $a_i$ and $b_j$ are atomic formulas or negations of atomic formulas. If variables are involved, they are supposed to be quantified universally over the whole formula. In fact, every statement from many-sorted predicate logic (with quantifiers) may easily be transformed to a set of statements of this format which is, in some sense, equivalent to the original statement (conjunctive normal form); such a transformation is described in books on AI, such as [14]. There are some syntactical restrictions imposed on the form of atomic formulas which appear in such rules: $a_1, \ldots, a_m$ are supposed to be written in the input and/or internal signature, whereas $b_1, \ldots, b_n$ are supposed to be written in the internal and/or output signature. The main two advantages of the use of a *rule-based specification formalism* are:

- it is a universal logical language that may be used to give a system description on an implementation independent conceptual level (also called semantic level, [21], or knowledge level, [20]),

- it is possible to define simple inference procedures for processing this type of formulas (simple types of resolution); if implementation is concerned, these inference procedures are supported by Expert System Shells.

Notice that the conventional predicate logic is a special case of the many-sorted logic; if additionally the arity of all the predicate symbols is zero and there are no constant and function symbols, we obtain propositional logic.

### 2.1.2 Meta-level aspects of a module

Apart from the object-level part, a module contains also a meta-level part in the form of a number of parameters and data on the state of the process in the module (describing a *meta-information state* of the module). A number of issues may be covered by this meta-information:

- *control parameters*

This involves information about the state of the process of the module (e.g., "has terminated successfully") or on how the inference process should be directed (e.g., information on the goals that are posed in goal-directed reasoning).

- *epistemic information*

This involves meta-information on the current (object-level) information state of the module; e.g., which object-level atoms are currently known and which are unknown.

- meta-information on the *history of the inference process* and *dependency links*;

This involves meta-information about the inference processes that have taken place: e.g., *truth maintenance aspects* like justifications of a given object-level fact, and the assumptions on which it is based.

The fact that the module stores two types of information (i.e., object-level and meta-level) results in multiple interaction types between modules. The following four types of interactions between modules may be distinguished:

| | |
|---|---|
| *object-interaction*: | from object-level to object-level |
| *meta-interaction*: | from meta-level to meta-level |
| *upward reflection*: | from meta-level to object-level |
| *downward reflection*: | from object-level to meta-level |

These four types enable us to give a strict definition of reflective reasoning patterns. This will be described in more detail in the next section on transformations.

A signature of the language in which meta-information states may be described is called a *meta-signature*. This signature may be built up from a number of generic (module and system independent) sort names, parameter names, and relation names; the object names are uniquely determined by the signature of the module and do not have to be specified separately. Some of the types of meta-information which were mentioned above are discussed in subsequent paragraphs; others are still being investigated.

#### *2.1.2.1 Control parameters*

The module itself is something what is static: it contains the knowledge and a collection of facts, but it does not contain any "meta-knowledge" about how to apply its knowledge to derive new facts. In fact, knowledge processing is done by an inference procedure, which determines *when* and *how* the rules of the knowledge base should be applied. Usually, such an inference procedure is equipped with a number of parameters which specify, for example, what type of conclusions we are interested in, which input atoms are especially important to be considered, how exhaustive the reasoning should be, etc. Roughly speaking, the inference procedure is a function, which takes as arguments a number of parameters, a module name, and an (object-level) information state, and returns a modified information state.

In this chapter we will give a general description of an inference procedure (we will call it Generic Inference Procedure, **GIP**).

The behaviour of **GIP** depends on the following parameters :

- *data set*,
  being a list of atoms over the input signature,
- *requestable set*,
  being a list of atoms over the input signature,
- *target set*,
  being a list of atoms over the output signature; these atoms are supposed to be split into three types of targets: **confirm, reject, determine,**
- *exhaustiveness*,
  which is expressed using the values **every, all_p, any.**

The meaning of these parameters is as follows. The *data set* specifies which input atoms should be taken as data for data-driven reasoning. The *target set* specifies which output atoms should be derived (i.e., found to be true), disproved (i.e., found to be false), or just determined (i.e., found to be true or false), according to whether these atoms are of one of the types **confirm, reject,** or **determine,** respectively. Specification of a target set is meant as posing a restriction on the set of output atoms we are interested in. If the target set is empty we interpret this as "no restriction": we are interested in every output atom (as if all possible output atoms belong to the target set).

The *exhaustiveness* determines how exhaustive the inference process should be: should it terminate when *any* goal from the target set has been achieved (meant as establishing a truth value), when *every* goal from the target set has been achieved, or when *all possible* (i.e., as much as possible) goals from the target set have been achieved, what means, that truth values of some atoms from the target set may still be **unknown,** but at the given situation it is impossible to find definite values for them (i.e., **true** or **false**).

The role of the *requestable set* is the following. According to the semantics of rules, a rule cannot be used if some of its conditions are **unknown.** On the other hand, it may be the case, that values of these conditions can be provided by another module, which, for certain reasons (e.g., efficiency) has been postponed. In such cases the system should be able to suspend the current module and to ask an appropriate question to another module. The role of the requestable set is to specify the set of atomic formulas which can be asked from outside. When such an atomic formula (possibly with variables) is encountered by **GIP**, the module becomes suspended (keeping the current line of reasoning), the proper query is generated and the control is passed to the supervisor. The supervisor activates the proper module (according to its backward rules; see below), and when an answer is generated, it is passed to the original module which then resumes its activity. Note that due to the possibility of making queries, atoms which are on the list **requestable** may also be put on the list **data**, although they are not known beforehand.

There are two points which should be discussed here. The first one is related to the notion of the *closed world assumption*. Many systems have the property, that if they cannot prove a given atom, then they conclude that the atom is **false**. In our case the value of such an atom is set to **unknown**, which seems to reflect more properly the ignorance of the system than setting the atom to **false**. On the other hand, if we want to have the "closed world assumption" in some modules of our system, we may need some more control parameters or structures for it (this may be investigated in the future).

The second issue concerns the status of the outcome of an activation of the module (i.e., calling the inference procedure). The inference procedure may end-up or suspend its activity; accordingly the module is in the state *terminated* or *temporarily suspended*. There are three different types of termination:

- *success*,
  when the global goal (specified by the target set and the exhaustiveness)
  has been achieved,

- *failure*,
  when the goal has not been achieved, but it may be achieved in the future (e.g., some atoms to be determined are still **unknown** ),
- *complete failure*,
  when the goal has not been achieved, and it cannot be achieved in the future (e.g., some atoms to be shown to be **true** have been shown to be **false**); possibly, after belief revision, this goal may become again "potential".

These three possible events are reflected in the meta-signature by the use of one special control predicate and three values, as is shown below. Let us note that the classical forms of reasoning are covered by **GIP** (data-driven reasoning when the target set is empty, goal-directed reasoning when the data set is empty).

There are also three types of situations in which a module is temporarily suspended, according to the exhaustiveness type of the query (**every, any, all_p**).

So far we have discussed the issue of *which parameters* should be given, when invoking the inference procedure. Now we will describe *how* to represent these control parameters and how some other control parameters are affected by the call of the inference strategy. The set of control parameters that refer to the specific object level language of a given module makes use of the module's input and output signatures. Let **IA** and **OA** denote the meta-language sorts for the names of all input, and output atoms of the object-level language, respectively (when there is no risk of confusion we will sometimes refer to atoms themselves instead of to their names). Control parameters are atoms of the meta-level language which are constructed from these names by use of the generic predicates: **data, requestable, target** (which can be viewed as meta-level input predicates), and **derived** (which may be viewed as a meta-level output predicate) and the values **confirm, reject, determine**; they belong to the meta-level sort **Target_types.** The meta-level sort **Exhaust_types** denotes the exhaustiveness types that may be assigned: **every, any, all_p.** The descriptions of some of the predicates are the following:

**data** on **IA**
  **data(a)**, where **a** is a name of an object-level input atom, states that atom **a** is in the **data** set of the module,

**requestable** on **IA**
  **requestable(a)**, where **a** is the name of an object-level input atom, states that atom **a** is in the **requestable** set of the module,

**target** on **OA x Target_types**
  **target(a, confirm)**, where **a** is the name of an output atom, states that atom **a** is in the **confirm** part of the target set,
  **target(a, reject)**, where **a** is the name of an output atom, states that atom **a** is in the **reject** part of the target set,
  **target(a, determine)**, where **a** is the name of an output atom, states that atom **a** is in the **determine** part of the target set.

**derived** on **OA x Target_types**
  **derived(a, confirm)**, where **a** is the name of an output atom, states that atom **a** has been confirmed during the last activation of the module,
  **derived(a, reject)**, where **a** is the name of an output atom, states that atom **a** has been rejected during the last activation of the module,
  **derived(a, determine)**, where **a** is the name of an output atom, states that atom **a** has been determined during the last activation of the module.

**to_query** on **IA x Target_types**
  **to_query(a, confirm)**, where **a** is the name of an input atom, states that atom **a** should be tried to be *confirmed* outside the module,

**to_query(a, reject)**, where **a** is the name of an output atom, states that atom **a** should be tried to be *rejected* outside the module.

**to_query(a, determine)**, where **a** is the name of an output atom, states that atom **a** should be tried to be *determined* outside the module.

The role of the **to_query** predicate is the following: when the inference procedure needs the truth value of a certain atom **a** from the requestable set, then such an atom is "marked" by assigning **true** to one of **to_query(a, confirm), to_query(a, reject), to_query(a, determine)**, according to whether **a** should be confirmed, rejected or just determined.

Notice that the first three predicates determine a setting of arguments of the inference procedure (and therefore we call them *input control parameters*), whereas the remaining two determine the (control) status of atoms which were affected by the call of the inference procedure (therefore we call them *output control parameters*). One might be surprised by the role of the output (control) parameter **derived**: it may seem, that its truth value may strictly be defined in terms of truth values of the original atoms. However, this is not the case. For example, an atom **a** which has been derived "in the past" may still be true, whereas its control counterpart, given by **derived(a, confirm)** may be false (in the current situation).

Furthermore, notice that the exhaustiveness type value may be "inherited" from the value of the general exhaustiveness parameter of the module (see also the discussion just below the **suspended** predicate).

Except of the control parameters that refer to the specific object-level language of the module, as listed above, there are some more parameters which are extensively used in the control of the module and the system. These parameters are given below. Here the meta-level sort **Termination_types** is used for the termination types, with object names **succeeded, fail, c-fail.**

**exhaustiveness** on **Exhaust_types**

This is an input control predicate of the module which determines the exhaustiveness of the inference process; e.g., **exhaustiveness(any)** indicates that the inference process would be stopped when any atom from the target set has been derived.

**termination** on **Termination_types**

This is an output control predicate which specifies the status of the result of the termination of the inference procedure call, becoming true on exactly one element of **succeeded, fail, c-fail** , according to this status; e.g., **termination(succeeded).**

**suspended** (0-ary predicate)

The meaning of this output control predicate is the following: **suspended** is true, if the module needs additional information (truth values) about certain atoms (that are described more specifically by the **to_query** predicate). If needed one could add one argument to this predicate denoting a global exhaustiveness type. In that case the supervisor can for instance receive the information that all possibilities (modules) should be tried. However it is not clear what the relation would be with the more specific **Exhaust_type**-assignments to atom types to be queried, that are not transferred to the supervisor. One solution could be to work for each module only with a global **Exhaust_type**; then the argument **Exhaust_type** may be omitted from the predicate **to_query** and added to **suspended** (and inherited from the module).

The output control predicates treated just above may serve as an input for the supervisor; because they are used by the supervisor in a context where other modules may play a role as well, they are then transformed to predicates with an additional argument referring to the module name

that is concerned. Thus the supervisor may make use of the following input control predicates (here **Modules** denotes the sort of all module names).

**termination** on **Modules x Termination_types**
**suspended** on **Modules**

These predicates may be used in the supervisor rules discussed below.

#### *2.1.2.2 Epistemic information*

Internal name links determine a correspondence relation between atoms of the object level and objects of the meta-level. Semantic links determine the truth values of (meta) predicates applied to (meta) objects, in terms of the truth values of (object level) atoms which are "name linked" to these (meta) objects. These dependencies can be specified by appropriate truth tables. An example is the meta-predicate **known** , defined on atoms; in this case the meta-predicate is defined by the truth-table in figure 1 below. A similar predicate may be defined on parameters **p**, indicating that there is a constant **c** such that **p = c** has been found to be true.

| a | true | false | unknown |
|---|---|---|---|
| KNOWN(a) | true | true | false |

Figure 1. The truth table of the meta predicate **known**

Such epistemic information of a module A may be used by another module B, in a way similar to that described in [23]. The transformation from A to B then involves transferring meta-information from one module A to object-information in another module B.

#### *2.1.2.3 Meta-information on the history of the inference process*

Here a whole book-keeping storage may be used on the dependency links between atoms (justifications, as in TMS, see [11]), as shown by the inferences that were performed until now. So, if a TMS for the module exists, this may be viewed as being included in the meta-level part of the module. Furthermore, the assumptions that were made may be stored, and for each atom a list of lists the assumptions on which it depends (as in ATMS, see [16]). This means that if an assumption is proposed for a module A from outside the module (for instance, by another assumption generating module B), this is transferred from the object-level of B to the meta-level of A. Inside the module A the step is taken to involve the assumption in the object-level reasoning. This subject will be investigated further in the future (in the context of a Modular Assumption-based Revision System).

## 2.2 Transformations and supervisor

### 2.2.1 Transformations

During the session information states are changing as successive modules are activated. To exchange information between the modules, some operations, which are called *transformations*, are used. The role of such a transformation is to determine which (output) atoms of one module correspond to which (input) atoms of another module and what the logical relations between these atoms are. Therefore, a transformation can be defined as a relation of four arguments - the first two specify an atom to be transformed and its truth value, the remaining two specify the resulting atom and its truth value. Obviously, this is a very general definition and says nothing about how to

specify transformations (the idea of specifying them by giving lists of all possible quadruples is not reasonable, e.g., because of the size of such lists). In practice however, transformations are well structured - atoms of similar structure (type) are transformed into atoms of another type, and the logical connections between atoms of both types can be uniformly specified by a truth table. This observation leads to the following definition of a transformation specification.

In DESIRE a transformation specification consists of:

- *signature of the domain* of the transformation, which determines the format of atoms to be transformed,
- *signature of the co-domain* of the transformation which determines the format of the resulting (transformed) atoms,
- *sort links*, that determine, which domain sorts should be mapped to which co-domain sorts. Such a link is a pair of sort names $<S_1, S_2>$ where $S_1$ and $S_2$ are sort names for the concerning signatures,
- *object links* that specify which object names of the domain signature correspond to which object names of the co-domain signature; these links are represented by ordered pairs $<n_1, n_2>$, where $n_1$ and $n_2$ are object names of the corresponding signatures; types of these objects should be consistent with sort links,
- *parameter links*, that specify which parameters of the domain signature correspond to which parameters from the co-domain signature; they are specified by pairs of the form $<p_1, p_2>$, where $p_1$ and $p_2$ are parameter names,
- *atom-type links* that specify which (domain) atom type corresponds to which (co-domain) atom type, and what is the logical relation between them. Such an atom-type link is specified by:
  - *type link* consisting of two atom types of the domain and co-domain signature, respectively; both predicates involved are assumed to have linked arguments for (possibly empty) subsets of their complete sets of arguments; these links are between arguments on linked sorts and are specified by using a convention to denote these linked variables, for instance: $<P(x{:}S_1, y{:}S_2, c), Q(d, y{:}S_3, x{:}S_4)>$ denotes that the first argument (variable denoted by $x{:}S_1$) of the first atom type is linked to the third argument (variable denoted by $x{:}S_4$) of the second atom type, et cetera (here c, d are some constants).
  - *truth table* which defines the logical relationship between atoms of both types; a truth table is a triple of the form $<<true, v_1>, <false, v_2>, <unknown, v_3>>$, where $v_1, v_2, v_3 \in \{true, false, unknown\}$.

The interpretation of the atom-type link which links atom type A to atom type B according to a truth table T is as follows: if A and B are closed atoms then the atom-type link simply states that the truth value of B can be determined from the truth value of A by the use of the truth table T, otherwise (i.e., some variables are involved) appropriate substitutions should be considered (variables in A are instantiated by object names, the corresponding variables of B are instantiated by object names determined by object links). For example, if we want to transform all instances $R(a, b)$ with constants a, b of the appropriate sorts $S_1, S_2$, to instances $S(c, d)$ where c is linked to b and d is linked to a, this can be specified by: $<R(x{:}S_1, y{:}S_2), S(y{:}S_3, x{:}S_4)>$. The truth table defines the semantic relations between these instances. In fact this is a way of saying that the symbols R and S in some sense denote the same relation.

The formalism described above enables us to identify easily the following four types of transformations: *object, meta, upward,* and *downward.* These types are determined according to whether the domain signature and co-domain signature of the transformation are object-level or meta-level:

| | | *co-domain* | |
|---|---|---|---|
| | | **object-level** | **meta-level** |
| *domain* | **object-level** | **object-interaction** | **downward reflection** |
| | **meta-level** | **upward reflection** | **meta-interaction** |

There is one very important type of use of transformations which should be discussed separately: transformations which play a role in, what is called in the literature, *belief revision* or *truth maintenance*. These transformations are still being investigated.

### 2.2.2 The supervisor

So far we have described the modules - devices which process information, and the transformations - devices which transfer information between modules. What is still missing is the *supervisor* - a device responsible for controlling the behaviour of the whole system. Its role is limited to:

- monitoring the current status of modules and inference procedure calls (has a given module already finished its job, has it derived anything, is it suspended, etc.),

- determining, on the basis of monitoring data, from which module data or meta-data should be transferred to which module, via which transformation, which module should be activated and which inference strategy should be applied (with which parameter setting).

The supervisor communicates with the rest of the system by means of "control parameters". On the one hand the supervisor can "read" values of some of these parameters (these values are provided to the supervisor by the inference procedures), and on the other hand, by modifying values of some other parameters the supervisor can cause activations of appropriate modules (calls of appropriate inference procedures with a proper setting of actual parameters). Moreover, it may use some internal parameters to represent some information about the state of the system. The knowledge of "when to do what" is encoded in rules. The rules are of the following types: *forward*, *neutral*, and *backward*.

The *forward rules* describe which module should be activated after another module is terminated, what data should be provided to it (i.e., from which module and via which transformation), and which setting of control parameters of inference strategy should be chosen.

The *backward rules* determine how to proceed when a module needs some extra information from other modules (i.e., when it is suspended). In such rules it is specified from which modules the information can be obtained, how such "source" modules should be activated, and which transformations should be used to transform the results to the suspended module.

The *neutral rules* allow the supervisor to reason about the current state of the system. Conclusions of these rules have no procedural meaning - they simply say in which point of the reasoning process the system is. These conclusions may trigger, however, some forward or backward rules.

The description given above gives an idea of how the supervisor acts and what is inside it, rather than how to specify it. Now we will show that the supervisor can be specified as any other module, i.e., by a list of signatures and rules. Given such a specification we can view it as a description of possible control and data flow of the whole system (declarative view), or as a kind of program which specifies which procedures should be called when (procedural view). The issue of how the

conclusions of the supervisor are used (e.g., for executing certain actions like activating a module or transforming results) is not fully discussed here.

The specification of the supervisor (or control structure of the system) consists, as in the case of any other module, of the three signatures and a knowledge base. First we describe the signatures.

The following sorts are common for input and output signatures of the supervisor:

Sorts:

**Modules** consisting of names of modules,
**Inference-procedures** consisting of names of inference procedures (**GIP** may be one of them).
**Transformations** consisting of names of transformations,
**Exhaust_types** consisting of some identifiers, including **all_p, every, any,**
**Termination_types: succeeded, fail, c-fail.**

Predicates:

**termination** on **Modules** × **Termination_types**
**suspended** on **Modules**

which belong to the input signature, and

**next-pre-trans** on **Modules** × **Modules** × **Transformations,**
**next-post-trans** on **Modules** × **Modules** × **Transformations,**
**nextmodule** on **Modules** × **Inference-procedures** × **Exhaust_types**

which belong to the output signature of the supervisor.

The internal signature is defined in an arbitrary way, according to what type of information should be represented within the supervisor to "make him aware" of what is going on. (If the control structure is simple (e.g., a cycle), then the internal signature may be empty; in case of a complicated system which may enter a number of specific phases, this signature may be quite extensive).

As said before, there might be three type of rules:

- forward, of the form:

**if** *condition* **then next-pre-trans(M,N,T) and nextmodule(N,IP,E),**

where *condition* is a conjunction of atoms over the internal signature and/or input signature (predicate **suspended** is not allowed, however),

- backward, of the form:

**if** *condition* **and suspended(M) then**
**next-pre-trans(M,N,T1) and nextmodule(N,IP,E) and next-post-trans(N,M,T2),**

where *condition* is a conjunction of atoms over the internal signature,

- neutral, which are ordinary rules written in the internal signature.

The intended meaning of output predicates is the following: **next-pre-trans(M,N,T)** should be read as: to transform atoms from module **M** to **N** before activation of **N**, transformation **T** is the proper transformation; similarly **next-post-trans(N,M,T)** should be read as: to transform atoms from module **N** to **M** after activation of **N**, transformation **T** is the proper transformation;

**nextmodule(N,IP,E)** should be read as: **N** is useful as the next module; activation of the module **N** can take place by calling the inference procedure **IP** with the exhaustiveness type **E**.

## 3 The notion of a Generic Task Model

The notion of a *generic task model* is relatively new and has not been formally defined in the existing literature yet. It originates from the notion of a *generic task*, and was identified by Chandrasekaran, [4, 5, 6]. In his papers he uses this term in an informal way, sometimes assigning different meanings to it. Recently, Treur made a first step towards a more precise definition of the concept of a generic task model, see [26]. The goal of this chapter is to present these notions in a more clear way.

The notion of a generic task, see [4, 6], originates from a detailed task-level analysis of reasoning patterns involved in the process of solving complex tasks. Complex reasoning tasks (like diagnosis, planning, design, etc.) occur in a similar manner in a number of application domains. They are called (complex) generic tasks and can be, in a quite natural manner, decomposed into a number of simpler subtasks. The process of decomposing tasks into simpler and simpler tasks leads to tasks which cannot be decomposed in a natural manner any further. In section 2.1 we called such tasks *primitive*. The idea is that these primitive tasks can be chosen in such a manner that their occurrence in a (complex) generic task depends on the task only and not on the application domain (of course the detailed knowledge involved may be domain-specific: for a given application domain an instantiation of such a primitive task is made). So, primitive tasks of the same type can often be identified in a whole variety of problems - therefore we call them *primitive generic tasks*. (Chandrasekaran used the term *generic task* for both complex generic tasks and primitive generic tasks, which may cause some confusion).

Usually, to solve a primitive task, only a restricted type of knowledge is needed (only the knowledge related to this specific sub-task), and a simple inference mechanism can be engaged. These tasks involve only certain types of knowledge and control regimes which are independent of the domain about which the reasoning takes place. The overall decomposition of the original task into a number of primitive subtasks, identification of types of knowledge relevant for these subtasks, identification of possible interaction types, and description of the dynamics of the whole process of solving this task, is called a *generic task* and a (formal) description of it is called a *generic task model*. Such a generic task model can be instantiated to a task model, by providing specific knowledge (which depends on the domain of concern) to the model. In other words, the task model is something what is close to a modular system in which all the modules are filled with specific knowledge, transformations are strictly defined, and the control structure is fully specified. The generic task model can be compared then to a task model (or a modular system), from which all domain-specific knowledge is deleted.

We will illustrate these ideas by a small example given in the Appendix (this example is a small fragment of a generic task model for diagnosis, SIX, described in [10, 28]).

The idea of splitting a complex task into subtasks has its counterpart at the level of specification. Namely, once a generic task model is given, it can be formally specified in terms of *generic modules*, *generic transformations*, and *generic control structure*. Such a "generic specification" can be called a *formalized generic task model*, or simply generic task model, when there is no risk of confusion. Now we describe these "generic" notions in more detail. It should be mentioned however, that both our specification framework for modular knowledge-based systems (DESIRE) and the issue of a formal specification of generic task models will be tested and evaluated further.

### 3.1 Generic module

A generic module is the counterpart of the notion of a primitive generic task. The specification of the generic module should include the following items:

- a *name* of the module; this name can be used during the specification of the control structure of the whole system,

- a *generic input signature* of the module, which specifies which domain-independent objects, relations, and functions are used in the input part of the information scheme. When the module is instantiated, the input signature will often be extended by adding names of domain specific objects,

- a *generic output signature* of the module, which specifies which domain-independent objects, relations, and functions are used in the output part of the information scheme. Again, when the module is instantiated the output signature will often be extended by adding names of domain specific objects,

- a *generic internal signature* of the module, which specifies which domain-independent objects, relations, and functions are used in the internal part of the information scheme. In this case, when the module is instantiated, the signature will often be extended by both names of objects and names of sorts, functions and relations, which are all domain specific. Of course there is the possibility that the internal signature of a module in a domain specific application is completely generic; in that case the module can be re-used in different application domains, as it is (*generic knowledge*).

To give an example of a generic module, let us think about a module which might be called ASSUMPTION SELECTION module. This module receives as an input some statements of the form **possible_assumption**(A), where A is a name of an atom of another module (e.g., the one which generates assumptions), and has to produce a statement of the form **best_assumption**(A). The input signature of our module contains then the name of the predicate **possible_assumption** and the output signature contains the name **best_assumption**. As an internal predicate we may have a binary relation called **better_than**. When the module is used in a system related to the transport domain, it can be instantiated by adding some (domain specific) rules of the form:

**better_assumption**('transportation_mode(train)','transportation_mode(car)'),

or

**if** **possible_assumption**('transportation_mode(train)')
**and** **possible_assumption**('transportation_mode(bus)')
**then** **best_assumption**('transportation_mode(train)').

### 3.2 Generic transformation

As we remember from the previous chapter, a transformation is supposed to translate information states of one (or more) module(s) into another information state. Moreover, it turned out that in the specification of transformations signatures play the crucial role: once the name- links and semantic-links have been specified on signatures, the transformation can be extended to all the atoms. Therefore in the case of generic transformations, only the output signature of the "source" module(s) and an input signature of the "target module" are relevant. Thus the specification of the generic transformation should include:

- a *name* of the transformation (to be used, for example, in the specification of the control structure of the system),

- a *signature of the arguments* of the transformation; usually, it will be a part of a generic output signature (or a list of output signatures) of the module (or modules) on which the transformation is supposed to act,

- a *signature of the co-domain* of the transformation; usually it will be a generic input signature (or a part of it) of a module to which the transformation is supposed to send data,

- a set of *name links* which link elements of both (generic) signatures,

- a set of *semantic links* (in the form of truth tables) which determine the meaning of the name-links.

If the generic task model is instantiated by domain knowledge, only the domain specific object links have to be specified in addition.

### 3.3 Generic control structure

In our approach there is no difference between the notion of the control structure and the generic control structure. The control structure describes the control flow in terms of truth values of some conditions. These conditions can be fully specified in terms of names of modules, names of transformations, "control predicates" discussed in chapter 2, and types of inference mechanisms. All these items are given in the specification of generic modules and generic transformations; if types of inference mechanisms are concerned, they usually involve only fragments of input/output signatures of the modules. In such a case, the generic control structure of the system can be specified in exactly the same way as the control structure of the "instantiated" system.

## 4 The role of a formalized generic task model in the development process

### 4.1 The added value of a formalized generic task model

In this section, we focus in particular on the added value of a *formalized* generic task model in the development process. The important role that may be played by an informal generic model, called interpretation model, is discussed in [3]. For instance, the analysis of verbal data of an expert protocol is very hard without having any global model for the reasoning process. Of course, creating or selecting the appropriate model may be a first bottle-neck. This may be supported by a library of existing task models, and some classificational structure for generic tasks. If a new generic task model is required, this may be designed as a configuration of generic modules of an available library of generic modules. Once this generation of a generic task model has been accomplished, much support will be provided to the knowledge acquisition process.

The extension we give to Breuker & Wielinga's approach (see [2]), is that instead of an interpretation model with informal semantics, we use the concept of formalized generic task model as defined in section 3. In the following main issues our approach has some added value:

- our formalized generic task models both have a *formal syntax* and a *formal semantics*; for instance, the former enables automated implementation generation, and the latter provides a basis for a foundational theory behind our work,

- in our generic task models, explicit and *detailed specifications* are given for the input and output data and for data-transformations defining *interactions* between the modules; this gives more support in the interpretation of these interactions than is obtained by simply saying that certain output data of some module are used (in some sense) as input data for another module,

- as a result of a more detailed specification, our notion of a generic task model makes *finer distinctions* between different generic tasks; for instance, if purely graphical representations of generic task models are drawn, where only (sketches of) modules and input and output links are displayed, two different generic task models may correspond to the same graph; in a formalized generic task model the semantics of the interactions between modules are also specified,

- our formal description of generic tasks enables one to identify and classify generic tasks in an exact manner: for instance it is possible to establish whether or not two given generic tasks are in fact the same generic task, or whether or not they belong to the same class of generic tasks,

- for any generic task model a generic but task-specific shell may be implemented and used in the development process, thus making the step to implementation much simpler.

In section 4.2 we will discuss how this added value of a formal notion of a generic task model can be exploited in knowledge acquisition.

## 4.2 Formalized generic task models in knowledge acquisition

We now come back to what was said in the introduction about the role of a formalized generic task model in the light of automated support to knowledge engineering. Above, following Breuker and Wielinga, it is argued that model-driven knowledge acquisition is a useful method for doing knowledge engineering, and that automated support for it is desirable. Of course to any informal notion of generic task or interpretation model, some implementation can be added. However, this implementation may have no one-to-one correspondence to the original informal generic task model; in any case it is hard to decide whether or not there is such a correspondence, since one tries to compare a formal object (the implementation) to an informal object (informal description of the generic task model that is concerned). As a result one may be inclined to keep the implementation for the precise definition of the concerning generic task model. Since the details of the chosen implementation language cannot be separated, this is conceptually unsound. The work of Chandrasekaran et al. (see [4]) tends in this direction by defining different types of task-specific knowledge representation languages: these languages are related to specific implementations, so, for example, the choice for object-oriented implementation languages is reflected in the description of the task-specific knowledge representation languages. As argued in Chapter 2, our specifications are on an implementation independent conceptual level (semantic level, [21]).

A formalized notion of a generic task model may be used as a basis for a generic implementation of the generic task model that has a one-to-one correspondence to the specification. This provides a generic *task-specific shell* for the generic task that is concerned. For building software tools to support model-driven knowledge acquisition, such a related implementation can be used as an environment that is offered to the knowledge engineer and may be used in sessions with the expert and with the enduser. These sessions may take place at every stage of the knowledge acquisition process. At early stages the generic task model serves as an empty (but task-specific) shell that gives structure to the overall reasoning process. Later on it may serve as a system design that is filled in with domain knowledge partially, and during sessions the expert may provide missing knowledge elements for each module. We mention the following possibilities:

- sessions where the *expert provides missing knowledge* for each module

It is possible to have sessions where the task-specific shell performs the reasoning process asking the expert for the missing domain specific knowledge at the moment it is needed in the reasoning. This assumes that the expert can express his or her knowledge, for instance in the form of rules or other logical relations. Some support for this can be added if for each module a list of relevant input data and a list of relevant output data is established beforehand (with the expert). During a session,

when a module is activated, these lists may be offered. Then for any given output datum that is needed, the expert may point (for instance by mouse) at the input data that are relevant for the concerning output datum, and also assign some requirements to them (e.g., positive or negative truth values). In cases that the expert still finds it hard to provide knowledge, this approach may be weakened to an approach where the expert only gives information on how input data and output data are linked for a number of cases:

- sessions where the *expert provides data on specific cases* for each module

If the expert is not able to provide knowledge for a certain module, another application can be made using such a task-specific shell by not asking the knowledge itself, but only the data (that may be viewed as consequences of the desired knowledge) in a specific case. The system may use inductive learning techniques to abstract the domain knowledge from these cases.

- sessions where *data on cases from another source* are analyzed for each module

It is also possible that it is not the expert who gives answers about these data, but some available database of cases.

- sessions with *enduser and expert* to obtain detailed requirements on functionality and modality

By involving both the expert and the enduser, it is possible to acquire knowledge about the interaction with the system as desired by the enduser. A task-specific shell may provide a structured environment to obtain detailed knowledge about this.

- sessions with the expert where *knowledge is mutually generated*

If the expert has provided some basic knowledge, the knowledge engineer may make some analysis of this knowledge. Based on this analysis (s)he may propose some (new) added knowledge, for instance of a strategic nature; for example this may involve the application of existing techniques for optimization (or OR-techniques). After adding this knowledge to the task-specific shell, sessions with the expert may be held to obtain his or her judgement (and possibly suggestions for modifications) on this added knowledge. This is an example of a more active role of the knowledge engineer in the knowledge acquisition process; in fact it is not knowledge acquisition in the strict sense, but rather mutual knowledge generation.

We will give a short illustration of how the (formalised) generic task model has influenced the knowledge acquisition phase during the design of a system for routing international blanco payment orders (the system has been built for Rabobank (one of the sponsors of the SKBS-A3 project), which provided domain experts and evaluated (very positively) the final implementation).

After a few half-a-day sessions with experts of Rabobank, an architecture of the task model has emerged (here we should mention our colleague, Pieter Geelen, who played a crucial role in the whole development path, and who finally implemented the system). Then, the first (draft) version of the generic task model has been formalized. This formal specification has been reviewed by experts (both from Rabobank and our group), which almost automatically lead to a detailed specification of the system (the target prototype in which about 80% of domain specific knowledge has been encoded). Maybe surprisingly, the Rabobank experts turned out to have no problems with reading the specification documents, and consequently they suggested a number of improvements and were able to provide domain knowledge in a "module-oriented" form (i.e., the provided rules could easily be translated into the language of appropriate modules). When confronted with the specification of the task model they become more aware of their own knowledge, of the role of particular subtasks, and of the dynamics of the overall process. Since that moment, filling in the model with specific rules was a matter of a few days. The final specification served as a solid basis for the implementation phase, which turned out to be an easy task. The system has been implemented in three ways: in the hybrid expert system shell Nexpert, in Pascal, and in Prolog.

The implementation of the system (maybe except the one in Pascal) is strongly related to the original specification, what makes the code very readable. Due to this, experts of Rabobank are convinced that they will be able to make the operational version of the system, as well as its updates, by themselves. For more details concerning the project see [13].

Another example of the use of the formalised generic task model is related to SIX (the model discussed in the Appendix). Originally designed as a task model of diagnosis, aimed at the medical domain, it has been successfully used during a design of a system in the domain of saving advices.

### 4.3 Further perspectives and relations to other work

There exists a number of techniques that may support knowledge acquisition. We discuss techniques of the following three kinds (following Motta et al. 1989, section 2.2, [17]).

Techniques of the first kind concern formal techniques that may be used in knowledge acquisition on a detailed level, such as multidimensional scaling, [22], repertory grid [1], and hierarchical clustering [15]. If more complex tasks are involved, the use of these techniques is restricted to some of the types of knowledge that are involved. Other types, like knowledge concerning procedural and supervising aspects of the task will have to be handled differently. For example, the design of a task model is not supported by these techniques. However, within such a task model, these techniques may be very useful to obtain more detailed knowledge.

A second kind of techniques used in knowledge acquisition is centred around a given (generic) task model and supports knowledge acquisition for applications that fit into this task model. Examples are MOLE, [12], based on Clancey's model of heuristic classification, [8]; also Chandrasekaran's work, [5,6], and KADS, [3], can be mentioned here. These techniques offer help in the model-driven (top-down) part of the knowledge acquisition process (see [17]), but in the (often earlier, bottom-up) phase where it is tried to establish a suitable task model there may remain a lack of support. Although some descriptions of task models are available, which are formal to a certain extent, the semantics of these models is not very clear: often mainly procedural descriptions are given (e.g., [9]). Since the task model here is considered an absolute framework for the chosen approach, researchers may not have much interest in providing a more clear semantics of it.

A third kind of techniques handles the creation, configuration, and tuning of a suitable task model to the task at hand. Being the more realistic case, where often an iterative alternation of bottom-up and top-down approaches is used, this is not an easy issue to address. Examples of some partial results in this area are PROTEGE, [18, 19] and KEATS-2, [17]. Here in fact the role of a task model is that of an object that is to be designed (or composed). So, semantically spoken, the perspective on a task model here differs considerably from the perspective on it in knowledge acquisition techniques of the second kind. To be able to support automatic handling of designs of task models in a flexible manner, a clear semantical description of these "design objects" is indispensable; the procedural descriptions mentioned above are not sufficient here.

The notion of a formalized generic task model as contributed by this paper may offer a clear and formal semantical basis to knowledge acquisition techniques of the second and third kind. Until now our own experience is mainly related to the use in knowledge acquisition techniques of the second kind (e.g., SIX, routing; see Appendix). Nevertheless, we expect that formal methods of specification as described here will have to play a major role in the process of development of knowledge acquisition techniques of the third kind. This will be investigated further in the near future.

## Acknowledgements

The research as reported here has taken place in the context of the research programme of our Artificial Intelligence Section. One of the aims of this programme is to develop methods, techniques and tools to support the design, specification, and implementation of modular knowledge-based systems, related to our DESIRE framework. The following persons and institutions (among others) have contributed to this research: Harm Brumsen, Pieter Geelen, Allard van Oostveen, Jack Pannekeet, Yao Hua Tan, and SKBS-project A3, IIICAD-project, Rabobank Nederland, Cap Gemini Pandata. Pieter van Langen has proof-read the final version of this paper.

## Appendix. An example of a formalized generic task model (specification)

We will illustrate the basic concepts of our framework by a small example (this example is a simplified version of a generic task model for diagnosis, SIX, described in [10], and [28]). When diagnosing, usually a number of *hypotheses* is possible and there is a number of possible *symptoms*, which can be observed by carrying out certain *tests*. The diagnostic process has to determine the order in which these tests should be performed in order to arrive at a definite conclusion (the truth of a certain hypothesis). This process can be split into the following subtasks:

- determine a hypothesis which, in a given situation, should be investigated first,
- determine a test which should be performed to (partially) confirm or reject the chosen hypothesis,
- perform the test and evaluate its results (is the symptom present or not)
- interpret the result of the test (how the result of the test affects the confirmed/rejected status of hypotheses),
- make a final conclusion (is the hypothesis confirmed or are further tests needed).

The very nature of these tasks determines which results of which subtasks are relevant for other subtasks; also the dynamics of the whole process can be easily identified (see figure 2).

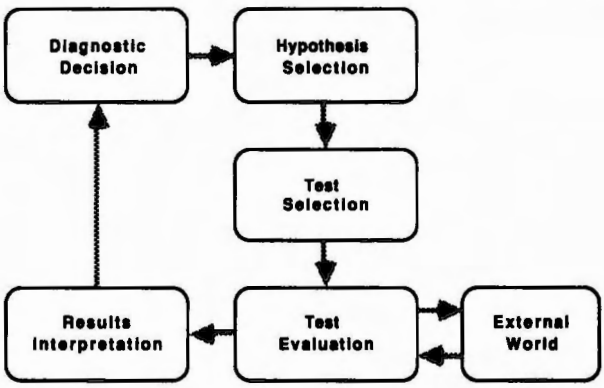



Figure 2. Diagnostic process: subtasks and interactions

Now we will formally specify the generic task model sketched above. The global architecture of the model is shown in the figure above, and a specification of all modules, transformations, and control structure is given below.

## A.1 Modules

**Hypothesis Selection Module**

The task of this module is to determine which hypothesis, in the given situation (process state), should be investigated first. The module gets as an input epistemic information about the evidence which has been gathered so far (which symptoms have been found to be present or not, which symptoms cannot be determined at all (e.g., due to the lack of proper equipment), which symptoms are still unknown, which hypotheses have been rejected). Applying its (domain-specific) knowledge it generates a conclusion of the form to_be_investigated(h). Formally, the module is specified as follows:

**module_name** hypothesis_selection
**sorts** Hypothesis_names, Symptom_names
**input**
present : Symptom_names
not_available : Symptom_names
rejected : Hypothesis_names
confirmed : Hypothesis_names
**output**
to_be_investigated : Hypothesis_names

**Test Selection module**

This module contains knowledge about (logical) relations between symptoms and hypotheses. When activated with "to_be_investigated(h)" in its information state, it will find a symptom (which presence should be confirmed or rejected) that is most relevant for determining h. Note that the input for this module should also contain epistemic information about symptoms. Formally:

**module_name** test_selection
**sorts** Hypothesis_names, Symptom_names
**input**
present : Symptom_names
not_available : Symptom_names
to_be_investigated : Hypothesis_names
**output**
needed : Symptom_names

Obviously, conclusions of this module (i.e., atoms of the form needed(s)) will be transformed by a downward transformation to atoms (of the meta-level part of the Test Evaluation module) of the form target(present(s), determine) (compare with Chapter 2).

**Test Evaluation module**

Usually, to verify the presence or the absence of a symptom, a number of measurements is needed and on the basis of these measurements some conclusions can be drawn. For example, to verify whether the patient has a fever or not, its temperature has to be measured, and according to the result an answer may be given (e.g., if the temperature is higher than 38° then the symptom "fever" is present). It may also happen, that the results of measurements (or their lack) will not lead to any conclusion concerning the given symptom - in such case the symptom is said to be not_available.

The task of the Test Evaluation module is ordering such measurements and analyzing their results. This module is specified as follows:

**module_name** test_evaluation_module
**sorts** Symptom_names, Parameter_names, Values
**input**
value_of : Parameter_names × Values
**output**
present : Symptom_names
not_available : Symptom_names

Note that the control status of atoms of the form value_of(parameter, value) will be (initially) set to *requestable*. When the module is activated it may happen that during an attempt to verify a statement of the form present(s), some atoms of the form value_of(parameter, value) (which may be unknown) will have to be asked. In such a situation, the module becomes suspended, the meta-level atom to_query(value_of(parameter, value), determine) will be set to true, and then transformed to the meta-level of the External World module as target(value_of(parameter,value), determine).

**The External World module**

This is not a real knowledge-based module. Its role is to provide specific information concerning various aspects of the object being diagnosed. We can think of this module as a user interface (the user is supposed to provide the information), a conventional data-base (the information is encoded in it), or an interface with another system. Although this module is not a typical reasoning module, our framework is expressive enough to specify it in terms of input and output signatures.

**module_name** external_world
**sorts** Parameter_names, Values
**output**
value_of : Parameter_names × Values

**Results Interpretation module**

When a new symptom is found to be present or absent, some hypotheses can be confirmed or rejected (on the basis of (domain specific) rules which express logical relations between symptoms and hypotheses). The Results Interpretation module performs this ordinary "update". It is specified as follows:

**module_name** results_interpretation
**sorts** Hypothesis_names, Symptom_names
**input**
present : Symptom_names
**output**
rejected : Hypothesis_names
confirmed : Hypothesis_names

**Diagnostic Decision module**

The role of this module is to determine whether the diagnostic process should be continued or not. The "not" decision may be taken in two cases: either a certain hypothesis has been found to be true (i.e., an atom of the form confirmed(h) has been found to be true), or the system was not able to confirm any hypothesis, even though it explored all possibilities. The specification of this module is as follows:

**module_name** diagnostic_decision
**sorts** Hypothesis_names
**input**
rejected : Hypothesis_names
confirmed : Hypothesis_names
**output**
continue (0-ary predicate)
unable_to_conclude_anything (0-ary predicate)
concluded : Hypothesis_names

## A.2 Transformations

Besides 7 transformations which are following arrows in figure 2, there are two more: the one from the Test Evaluation module to the Test Selection module (the later module must be informed about what is already known about symptoms), and the one from the Results Interpretation module to the Hypothesis Selection module (if needed, rejected hypotheses should be transferred to the later module). In most cases, transformations are identity functions on appropriate signatures (modulo appropriate renamings). One exception is the transformation from the Test Evaluation module to the External World module (the "ask_question" transformation), and its counterpart (the "transfer_answer" transformation - a transformation which passes results of the External World module back to the Test Evaluation module). We will specify only the first one. This transformation is supposed to transform any "question" (of the Test Evaluation module) of the form value_of(parameter, X) (which is reflected on the meta-level as to_query(value_of(parameter, X), determine), to an atom target(value_of(parameter_name, X), determine) of the meta-level part of the External World module. This transformation can be specified as follows:

**transformation_name** ask_question
**domain**
**sorts** Parameter_names1, Values1
**predicates**
value_of : Parameter_names1 × Values1
**co_domain**
**sorts** Parameter_names2
**predicates**
value_of : Parameter_names2 × Values2
**sort links**
<Parameter_names1, Parameter_names2>
<Values1, Values2>
**atom_type_links**
**type_link**
<to_query(value_of(X:Parameter_names1, Y:Values1), determine),
target(value_of(X:Parameter_names2, Y:Values2), determine)>
**truth_table** <<true, true>, <false, false>, <unknown, unknown>>

Let us note, that the identifier Parameter_names1 refers to the sort Parameter_names of the Test Evaluation module, whereas the identifier Parameter_names2 refers to the sort Parameter_names of the External World module (in practice both sorts are represented by the same set of names). The same holds for Values.

### A.3 The Supervisor

The control structure of the system, partially illustrated in figure 2, is specified by a number of rules. We will concentrate on the most interesting fragment of the system i.e., the interaction between the Test Evaluation and the External World modules. The former one is activated when the Test Selection module successfully selects a test (tests):

```
if termination(test_selection, succeeded) then
    next-pre-trans(test_selection, test_evaluation, id) and
    nextmodule(test_evaluation, GIP, any).
```

When the Test Evaluation module is active, it may happen that certain data will have to be requested from the External World module. In such case the Test Evaluation module will become suspended, a proper query will be transferred (by the ask_question transformation specified above) to the External World module , which will become active, and finally results will be transferred back to the Test Evaluation module, which will then be resumed.

```
if suspended(test_evaluation) then
    next-pre-trans(test_evaluation, external_world, ask_question) and
    nextmodule(external_world, GIP, all_p) and
    next-post-trans(external_world, test_evaluation, transfer_answer).
```

## References


[1] Boose, J.H., *A knowledge acquisition program for expert systems based on personal construct theory*, Int. J. Man-Machine Studies 23 (1985)

[2] Wielinga, B.J. and Breuker, J.A., *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, pp 17-44.

[3] Breuker, J.A., et al., *Model-driven knowledge acquisition: interpretation models*, Deliverable task A1, ESPRIT Project 1098, SWI, University of Amsterdam, Holland, 1987.

[4] Brown, D.C., Chandrasakaran, B., *Design problem solving. Knowledge structures and control strategies*, Research Notes in Artificial Intelligence, Pitman, London, England, 1989.

[5] Chandrasekaran, B., *Generic tasks in knowledge-based reasoning: high level building blocks for expert system design*, IEEE Expert 1 (13), pp. 23-30, 1986.

[6] Chandrasakaran, B., *Towards a functional architecture for intelligence based on generic information processing tasks*, Proceedings of IJCAI-87, Milano, Italy, pp 1183-1192.

[7] Chang, C.C., Keisler, H.J., *Model Theory*, North-Holland, 1977.

[8] Clancey, W.J., *Heuristic classification*, Aritficial Intelligence 27 (1985), pp. 215-251.

[9] Clancey, W.J., *Representing control knowledge as abstract tasks and metarules, in Expert systems applications*, Bolc,L., Coombs, M.J. (eds), Springer Verlag, 1988.

[10] Dam, G., *A design for a strategic interactive diagnostic expert system*, Report PE8901, Programming Research Group, University of Amsterdam, Holland, 1989.

[11] Doyle, J., *A Truth-Maintenance System*, Artificial Intelligence 12 (1979), pp. 231-272.

[12] Eshelman, L., McDermott, J., MOLE: *A knowledge acquisition tool that uses its head*, Proc. AAAI '86, Philadelphia, PA, 1986

[13] Geelen, P.A., Kowalczyk, W., *A knowledge-based system for routing international blanco payment orders*, Proceedings AIT '90 (in Dutch).

[14] Genesereth, M.R., Nilsson, N.A., *Logical foundations of Artificial Intelligence*, Morgan Kaufmann, 1987.

[15] Johnson, S.C., *Hierarchical clustering schemes*, Psychometrika 32 (1967)

[16] deKleer, J., *An Assumption-Based TMS*, Artificial Intelligence 28, (1986), pp. 127-162.

[17] Motta, E., Rajan, T., Eisenstadt, M., *Knowledge acquisition as a process of model refinement*, Open University, Human Cognition Research Lab., Techn. Rep. HCRL-40, 1989, Milton Keynes, U.K., also in: Int. J. of Man-Machine Studies (1989)

[18] Musen, M.A., *Generation of model-based knowledge-acquisition tools for clinical-trial advice systems*, PhD thesis, Stanford University, 1988

[19] Musen, M.A., *An editor for the conceptual models of interactive knowledge-acquisition tools*, Proc. of the Third Knowledge Acquisition for knowledge-based systems Workshop, Banff., 1988

[20] Newell, A., *The knowledge level*, Artificial Intelligence 18 (1982), p 87-127

[21] Pylyshyn, Z.W., *Computing in Cognitive Science*, p. 49-92 in: Posner, M.I., (ed.), Foundations of Cognitive Science, MIT Press, Cambridge Mass. 1989

[22] Shepard, R.N., Romney, A.K., Nerlove, S.B., *Multidimensional scaling: Theory and applications in the behavioural sciences*, Vol. 1, Seminar Press, New York, 1972

[23] Tan Y.H., Treur, J., *A logical description of a bi-modular system for non-monotonic reasoning with dynamic assumptions*, Report PE8907, Programming Research Group, University of Amsterdam, 1989.

[24] Treur, J., *Completeness and definability in diagnostic expert systems*, Proc. Eur. Conf. on AI , ECAI-88, München, 619-624, 1988.

[25] Treur, J., *On the use of reflection principles in modelling complex reasoning*, Report P8812, Programming Research Group, University of Amsterdam, 1988. (to appear in: Int. Journal of Intelligent Systems 5, 1990.)

[26] Treur, J., *Design of modular and interactive knowledge-based systems*, Report P8826, Programming Research Group, University of Amsterdam, 1988.

[27] Treur, J., *A logical analysis of design tasks for expert systems*, Int. Journal of Expert Systems 2 (1989), p. 233-253

[28] Treur, J., *Incompleteness in domains with a layered empirical basis and heuristic reasoning in SIX*, Report PE8903, Programming Research Group, Expert Systems Section, University of Amsterdam, 1989. (short version in: Proc. NAIC-89)

[29] Treur, J., *Description levels and interaction types in modular knowledge-based systems*, Report PE8906, University of Amsterdam, Programming Research Group, Expert Systems Section, 1989.

[30] Wielinga, B.J., Bredeweg, B., and Breuker, J.A., *Knowledge Acquisition for Expert Systems*, Proc. ACAI-88, pp. 96-123, 1988

# ALTO: An Automated Laddering Tool

Nigel Major and Han Reichgelt*
Artificial Intelligence Group
Department of Psychology
University of Nottingham
Nottingham NG7 2RD
England
email: nigel@psyc.nott.ac.uk
email: han@psyc.nott.ac.uk

**Abstract**

This paper presents ALTO, an Automated Laddering Tool. ALTO is an implementation of the laddered grid knowledge elicitation technique. The work reported here has been carried out as part of more general research towards building a fully integrated knowledge engineering workbench. In this paper, we first introduce the technique of laddering and then describe our implementation. One of the more interesting aspects of ALTO is its ability to analyze the knowledge that has been elicited. We also discuss some of the more general implications that our work on ALTO has thrown up for the design of a knowledge engineering workbench.

## 1 Introduction

One of the most difficult aspects of the job of a knowledge engineer is the task of eliciting knowledge from an expert. This task is particularly difficult because experts find it hard to formulate their knowledge explicitly. To combat this, a number of techniques have been devised to maximize the yield of usable knowledge obtained from experts. Broadly, these fall into two categories, standard techniques and contrived techniques. Standard techniques involve interviews, either structured (Hart, 1986, Grover, 1983) or unstructured (Weiss and Kulikowski, 1984), or descriptions

*The research reported here was performed as part of the Esprit-funded ACKnowledge project. Colin Corbridge, Peter McGeorge and Gordon Rugg informally evaluated an earlier version of ALTO, and provided valuable feedback which has substantially improved the tool. We are also grateful to Mike Burton for helpful comments.

of the experts task, produced either by the expert as the task is performed, or by one expert watching another (Johnson *et al.*, 1987, Clarke, 1987).

Whereas standard techniques tend to be general purpose, contrived techniques have been developed with particular goals in mind. For example, a contrived technique may be aimed at eliciting a certain type of heuristic knowledge or may have been designed with a view to performing certain statistical analyzes on the knowledge. Examples of contrived techniques include different types of sorting (Gammack, 1987a, 1987b), repertory grid analysis (Shaw and Gaines, 1987) and the limited information task (Hoffman, 1987). Laddering is another example. This paper describes, ALTO, an automated laddering tool, written in Common Lisp.

ALTO was built as part of a larger effort to build a knowledge engineering workbench (KEW), whose aim is to integrate a number of different elicitation and machine learning tools in a single environment. The knowledge engineer is expected to use these different acquisition tools to produce a common knowledge base. In addition, KEW is expected to give the user advice and guidance on when to use which tool. The hope is that by making available a large number of different tools, the process of building a knowledge base is facilitated and hence becomes more efficient (Marty *et al.*, 1989).

The organization of the paper is as follows. In section 2, we discuss laddering in more detail. In section 3, we describe the current state of the laddering tool. In section 4, we give an example of ALTO in operation. In section 5, we discuss some of the implications that our work has for the design of a KEW. Section 6 finally concludes and discusses further work.

## 2 The laddering technique

In laddering, the expert and the knowledge engineer construct a graphical representation of the application domain in terms of the relations between domain elements (Rugg *et al.*, 1990). A laddering session is somewhat like a structured interview with the elicitor giving one of a standard set of prompts to the expert. The result is often represented in a semantic network type representation with nodes connected by labelled arcs.

Depending on the type of knowledge that the knowledge engineer tries to elicit, one can distinguish between what one might call "task-oriented" and "domain-oriented" laddering. In task-oriented laddering, one tries to get at the knowledge that an expert has of the structure of the task at hand. The arcs in the semantic network therefore typically include "subtask" and "supertask". In "domain-oriented" laddering, the emphasis is on what Clancey (1983) calls "structural knowledge", knowledge about the types of entity that are used in a domain. ALTO is restricted to domain-oriented laddering. In the sequel, whenever we speak of laddering, we mean "domain-oriented" laddering.

In using the technique the elicitor enters the conceptual map of the expert

at some point and then attempts to move around it with the expert. One must start the expert at one "place" in the domain and try to move up and down levels in it. Typically, laddering involves the following three steps:

*Step1:* Ask the expert to generate a starting point (a seed item) and start him or her off with this seed item.

*Step2:* Move around the domain map using the following prompts

1. To move DOWN the expert's domain knowledge:
   - Can you give examples of ITEM ?
2. To move ACROSS the expert's domain knowledge:
   - What alternative examples of CLASS are there to ITEM ?
3. To move UP the expert's domain knowledge:
   - What have SAME LEVEL ITEMS got in common ?
   - What are SAME LEVEL ITEMS examples of ?
   - What is the key difference between ITEM 1 and ITEM 2 ?

*Step 3:* Elicit attributes for the various concepts.

The knowledge engineer may move around the domain map of knowledge in any order which seems convenient. As the elicitation session progresses, the engineer keeps track of the elicited knowledge by drawing up a network on a large piece of paper. This representation allows the knowledge engineer to make decisions (or ask questions) about what constitute higher or lower order elements in the domain.

Laddering has been shown to be a powerful technique in classification domains (Burton *et al.*, 1988) with a high productivity of knowledge elicited for a given time length of elicitation. However, as with other contrived techniques, experts are often somewhat reluctant to use laddering (Schweikert *et al.*, 1987).

ALTO is not the first implementation of a laddering tool. Both ETS (Boose, 1985) and its successor AQUINAS (Boose and Bradshaw, 1987) include a laddering tool. This tool relies on Hinkle's (1965) analysis of this technique. As in ALTO, the ETS tool is restricted to domain-oriented laddering. However, there are two important differences. First, the ETS tool uses the prompts "why" and "how" to move up and down the hierarchy respectively. As a result, the concepts elicited are attributes of objects. Examples are concepts like "loving", "warm" etc. The hierarchical relationship between these concepts can then be seen as follows: if $X$ is a superordinate concept to $Y$, then every object that has property $Y$ also has property $X$. Thus, if "loving" is a superordinate concept of "warm", every person who is warm is also loving. ALTO's domain-oriented laddering, on the other hand, is geared towards eliciting the actual domain objects (or classes of such objects). Thus, if $X$ is a superordinate concept of $Y$, then this is to be interpreted as "a $Y$ is

a $X$" or "every $Y$ is a $X$". For example, "car" is a superordinate concept of "bmw" because a bmw is a car. Although ALTO allows one to specify attributes of the different concepts, no attempt will be made to establish any relationships between the attributes. A second difference is that ALTO is able to analyze the elicited knowledge, and to use this analysis to drive further elicitation, as we shall show in section 3.3.

# 3 ALTO

ALTO was originally implemented in Poplog CommonLisp, with the user interface being implemented in Sunview using GEC's UI-Tool, and has recently been re-implemented in Sun Common Lisp with PCE. It carries out the elicitation in a system-driven fashion, in the manner of a knowledge engineer and expert performing the same task on paper. The computer asks questions and gives comments in a text window and the expert types in domain terms in response to these questions. ALTO has a number of settings which determine the degree to which the user can control the laddering process. These settings offer a choice of laddering experience and laddering strategy. Experience can be set to novice or expert. When experience is set to novice, ALTO will decide what to do next; when it is set to expert, the user can control the behaviour of the system, and decide what to do next, through a panel of control buttons. Strategy is either depth-first or breadth-first and guides the way in which the hierarchy is built up. Thus, when a depth-first strategy is used the system will insist in laddering down from a newly created node until the user indicates that a leaf-node is reached. We discuss the different settings in more detail in section 3.2.

The control buttons allow the user to ladder up, ladder down, give attributes for a concept, or to analyze the knowledge base. The ladder up and ladder down facilities are very much as one would expect them to be, given the description of the laddered grid technique in the previous section. The knowledge analysis feature is discussed in section 3.3. When the user selects the option of giving attributes for a concept, ALTO will first ask which concept the attributes are to be defined for, and then ask for an attribute and its value. When laddering experience is set to novice, ALTO will then ask for any other concepts with this attribute. If the user specifies any such concepts, and the concept is already known to the system, ALTO will simply associate this attribute and a value to be determined from the user with the concept in question. If the concept is not known, then ALTO will first interact with the user to determine the appropriate place of the new concept in the concept hierarchy. If the user specifies an attribute value pair that is already true for an object because of inheritance, ALTO will generate a warning, and ask the user if they want to persist in doing so. The various options are illustrated in section 4.

As the domain is built up, a graphical representation of the structure of the knowledge base is displayed in the concept hierarchy window. This window can

be directly operated on independently of the tool. This allows the expert to view concepts, delete them, or edit them. He or she can also rename concepts or swap them. The ability to act on the knowledge base allows the expert to enter details about the domain either using the tool or, if he of she is more confident, by doing the same thing directly. If the user wants to see the information about some concept in the knowledge base, a separate window appears in which all the attributes of the concept in question are shown. Figure 1 shows the interface as seen by the knowledge engineer, working on the domain of flint classification.

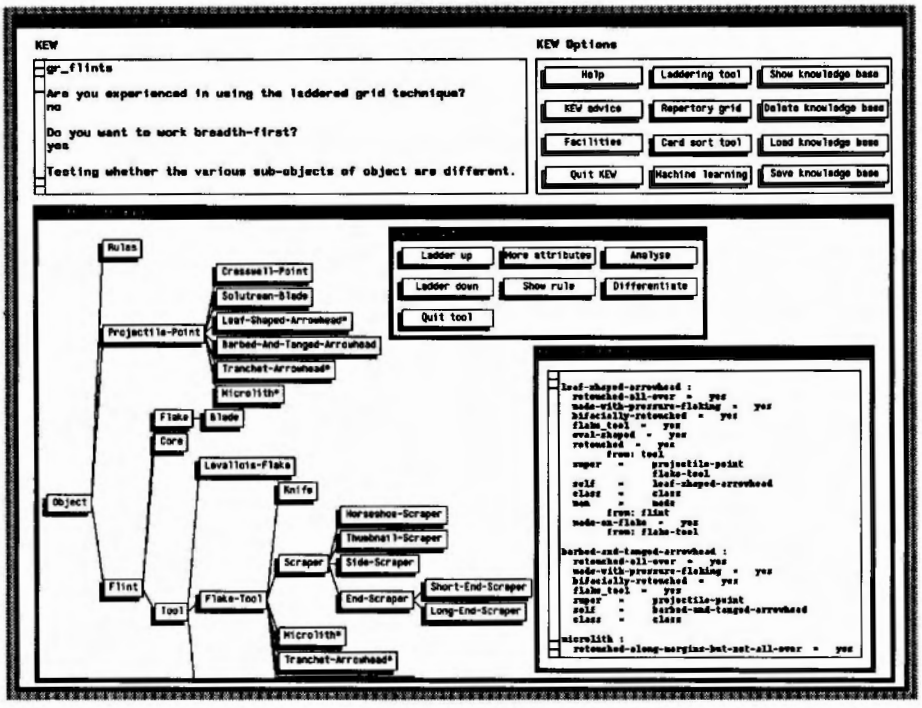



Figure 1: The laddering tool working on the flint classification domain

## 3.1 Knowledge representation

ALTO uses CommonSloop as its basic knowledge representation language. CommonSloop is a simple object-oriented language, which was originally written in FranzLisp by Peter Jackson and later ported to Common Lisp and improved by Han Reichgelt (Reichgelt, 1987). Each concept is represented simply as a set of slots, most of which are user-defined and represent properties of the concept. However, in addition there are a number of system-defined slots, which are mainly used

for book-keeping information. For example, CommonSloop stores a pointer to the parent(s) of a concept in its `parent` slot. CommonSloop assumes that the topmost concept in the concept hierarchy is called `object`, and ALTO shares this assumption.

CommonSloop allows multiple inheritance. In the interface, this is represented by drawing the child twice rather than by tangled graphs. The reason is that it is common for application domains to consist of large numbers of classes, many of which have more than one parent. Tangled graphs would soon become very difficult to understand in such domains. Although in ALTO the child is drawn twice, it is treated as exactly the same concept. Any subclasses created below it will therefore be seen in both areas of the domain in which it occurs. In order to indicate that we are dealing with a child which has multiple parents an asterisk is displayed behind any such child in the concept hierarchy window (e.g. the concept Microlith in figure 1).

In CommonSloop, as in virtually all other object-oriented representations, each object has a meta-class associated with it, although we hide this feature from the user of ALTO. Meta-classes can be used to define methods that apply to the object as an object, rather than as a representation of a class of entities in the world. Thus, a method, such as `calculate-speed` would be associated with the object `car` directly, whereas a method such as `sub`, which determines how to go about making sub-objects (children) of `car`, would be defined in its associated meta-class. By default, the meta-class associated with an object in the object hierarchy is the meta-class `meta-class`.

The fact that each object has a meta-class associated with it was used extensively in the implementation of ALTO. After all, most laddering operations, such as defining a new parent or defining a new sub-object, are applied to an object as an object in its own right, rather than as a representative of a class of entities in the world, and are therefore most properly seen as meta-operations. As a consequence, the interface between the laddering routines and CommonSloop has to a large extent been implemented through the addition of methods to `meta-class`.

The version of CommonSloop that we have used in ALTO supports default inheritance. Thus, attributes associated with higher-level concepts can be overridden in their children. This choice has a number of implications for the ease with which ALTO can be integrated in a larger KEW. We return to these in section 5.

## 3.2 Differences in user requirements

In the initial prototype of ALTO, the system itself was generally in control. For example, every time a new concept was created, the system would ask the user if he or she wanted to give any attributes for this concept, and if so, which ones. The decision to let the system maintain strict control was made because the majority of users, whether they were knowledge engineers or domain experts, would not be familiar with the use of the laddering technique. However, as a result of the evaluation of this prototype, it emerged that knowledge engineers who were familiar

with the technique found this style too constricting. So the second prototype has been designed to cater for both categories of user. When starting up the tool, the system asks the user if he or she is familiar with the technique or not. If not, then the dialogue is system-led as before. If the user is familiar with the technique then the responsibility for driving the interaction is passed to the user who must decide when to elicit attributes, or new classes, or when to use knowledge analysis facilities.

A second inflexibility in the initial prototype was that the system would ladder in a depth-first manner, and would insist on continuing the direction of the laddering until either a leaf of the concept hierarchy was reached, or the top. Thus, once the knowledge engineer had decided he wanted to ladder down, the system would insist on laddering down until a leaf was reached. However, evaluation again suggested that knowledge engineers often use laddering in a breadth-first manner, first getting all subclasses of a concept before laddering down on one of the subclasses. This issue has been dealt with in the second prototype, which therefore allows users to decide which strategy they prefer to follow.

Again, the issue of degree of user control has important implications for the design of a larger KEW. We return to these in section 5.

## 3.3 Knowledge analysis

In addition to the standard facilities that any automated laddering tool should provide, ALTO has an additional feature which makes it more than just a implementation of the paper-and-pencil laddering technique. ALTO is able to analyze the knowledge contained in its knowledge base, and to use this analysis as a basis for further elicitation. When the user selects the analysis option, ALTO will first ask which concept the user wants to analyze (the default is the topmost object `object` in the concept hierarchy). Knowledge analysis takes place in two phases. In the first phase, ALTO will determine whether any two children of a concept have distinguishing attributes. If two concepts have the same superclass and attributes, ALTO spots that the user has made an implicit distinction (they are different concepts) but has not expressed that difference in terms of attributes. The expert is then asked if he or she wants to differentiate between the concepts, and if so, to give an attribute which either pertains to only one of the concepts, or for which the two concepts have different values. The new attribute may of course itself be suggestive of areas of the domain so far untouched by the elicitation; there may be a whole class of concepts, so far unelicited, which also have this attribute. Depending on its setting, ALTO will check with the user whether this is the case.

The second phase of knowledge analysis involves checking whether all subclasses of a concept have an attribute and a value in common. If this is the case, then it might be more appropriate to associate the attribute and its value with the higher-level concept. ALTO will check this suggestion with the user, and generalize the attribute to the parent concept if the user indicates that this should be done.

There are a few remarks to be made about these knowledge analysis features

of ALTO. First, ALTO's knowledge analysis is in a sense independent of the fact that ALTO is a laddering tool. It can be applied to any tool which delivers output in an object-oriented language. In fact, although we originally implemented the knowledge analysis feature as a set of Lisp functions, we have recently re-implemented it by associating a number of methods with the meta-class associated with each concept in the concept hierarchy. This is a powerful feature, as it means that the knowledge analysis is no longer bound to ALTO, but can be used by any tool which has Sloop as its knowledge representation.

Second, since the version of CommonSloop that we use in ALTO supports default inheritance, we could make the test in the generalization phase less strict. Rather than suggesting that an attribute be generalized only if all children of a concept have the same value for this attribute, ALTO could suggest the attribute for generalization if most children share the same attribute value pair. How many "most children" should be is of course an open question. We return to the notion of default inheritance and the dangers inherent in it in section 5.

## 3.4 Evaluation

The tool has been informally evaluated by asking a small number of knowledge engineers some of whom are experienced in the use of laddering, and some of whom have never used it before, to use ALTO. This raised a number of shortcomings many of which have been put right in a second implementation of ALTO. Some of these have been discussed in section 3.2. For example, ALTO now supports a number of different settings which allow the user to change the degree of control that ALTO assumes during an elicitation session, and the way in which object hierarchies are to built.

A further lesson to come out of the informal evaluation of the first prototype of ALTO was that keyboard interaction and mouse interaction should be kept separate. Without this, the hand was constantly moving from one to the other. For example, the first prototype popped up a menu with yes and no as options whenever a yes or no question was asked. This was changed so that the keyboard could be used to answer such questions; this is more suitable as the keyboard would already be in use for entering domain information when the question was asked.

A number of possible extensions remain. For example, some indication of the amount of information associated with objects in the knowledge base should be shown when displaying the knowledge base. Another possible extension would be to allow the user to develop different hierarchies during various interactions with ALTO. ALTO should then provide support in the integration of them.

At the moment, some of our colleagues are running a more systematic experiment in which they compare the use of ALTO (without using knowledge analysis) with the paper-and-pencil use of laddering. However, it is too early to say what the results of this evaluation will be, as the experiments are still in progress.

# 4 Example of ALTO in use

To give the reader a flavour of ALTO and its knowledge analysis capabilities, we give a short example. In our example we have moved from the domain of flint classification to cars in order to make the elicited knowledge more familiar. It should be borne in mind that a lot of the interaction of ALTO with the user is done by direct manipulation which we do not attempt to represent here.

On starting up, ALTO asks the user whether they have experience of laddering and whether they would prefer to work depth-first or breadth-first. Let us assume that the user is a novice and wants to work breadth-first.

The user starts laddering down from `object` (the top-level object) and is asked to provide a subclass. The user specifies `car`. Because ALTO is running in the novice setting, it will then immediately ask the user to specify attributes for this new object. ALTO would have let experts decide for themselves when they wanted to specify attributes. However, the user declines.

Since ALTO is using a breadth-first strategy, it will now ask the user whether they want to include any other subclass of `object`. If ALTO had been using a depth-first strategy, it would have tried to ladder down from `car` until it reached leaf nodes and then returned to determine whether `object` has any other (immediate) subclasses. The user types in `italian product`. As before, he or she is asked whether they want to specify any attributes of this new object. The user decides to decline to do so for the time being. ALTO asks the user if they want to give any further subclasses of `object`, but the user declines.

ALTO will now return to the `car` and ask the user if they want to give subclasses for this object. The user specifies `bmw` without any attributes for this object, and `alfa romeo` with the attribute `wheels` and value `4`. Because of its novice setting, ALTO asks the user if they want to specify any other objects with the attribute `wheels`, thus using the attribute to attempt further elicitation. The user replies `bmw3`. As this object is not yet in the hierarchy, ALTO asks which class it is a subclass of. The user answers `bmw` and also gives it the value `4` as the value of its `wheels` attribute. Again ALTO asks if any other class has the `wheels` attribute. The user says that none have.

ALTO now returns to the class `car`, and asks the user if they want to specify any other subobject. The user declines. ALTO now asks the user if they want to give any subobjects of `italian product`. The user accepts this invitation and specifies `gucci shoes` and `alfa romeo`. Note than the latter object now has two parents. The user does not specify any attributes for either.

ALTO continues from the newly created objects, and asks the user if he wants to ladder down from `bmw`. The user takes up this invitation and specifies `bmw5` as a subclass, with attribute `wheels` and value `4`. The user also decides not to ladder down from `alfa romeo` and `gucci shoes`. ALTO will now ask the user whether they want to ladder down from `bmw3`. The user wants to do so, and specifies `bmw325M`. Asked to specify attributes for the new object, he specifies `wheels` with

value 4. ALTO warns him that this attribute is already true of this object, as its inherits it from bmw3. The user therefore declines to associate this value attribute-pair with this object. Laddering now stops: the user decides not to ladder down from bmw5 and bmw325M. Thus, we have the following hierarchy and objects as specified in figure 2, where as in figure 1, an asterisk indicates that an object has multiple parents.

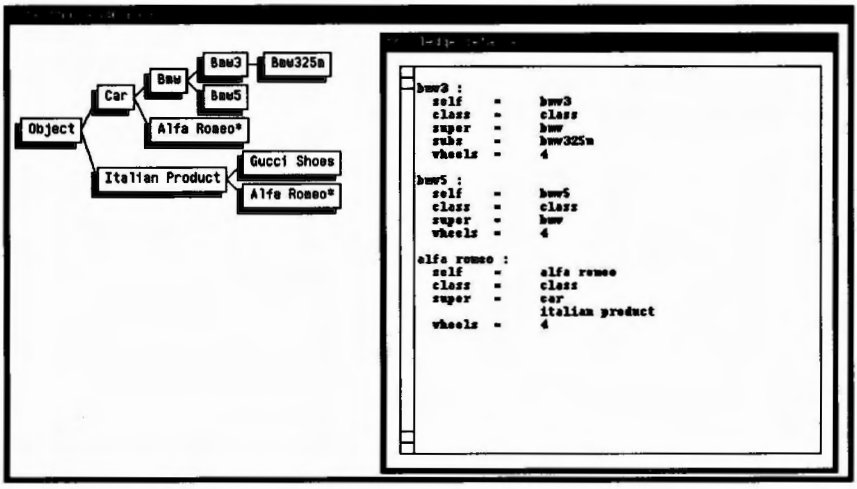



Figure 2: The initial hierarchy in the car domain

The user, now happy with his hierarchy, decides to analyze it. ALTO asks if any particular class should be analyzed and is told to analyze the whole hierarchy. As we said in section 3.3, analysis takes place in two phases. During the initial phase, ALTO will try to establish whether the various children of a concept have different attributes. Since the children of `object`, `car` and `italian product`, have no distinguishing attributes, the user is first asked to differentiate between the objects `car` and `italian product`. After all, there is an implicit difference due to the structure of the hierarchy but no explicit difference in terms of attributes. The user associates the attribute `appearance` with `italian product` with value `elegant`. The user is asked whether he wants to specify any other objects with attribute `appearance` but declines. Analysis now continues and the user is asked to differentiate between `bmw3` and `bmw5`, but declines to do so. Since all other children of the same concept have different attributes, the first phase of analysis finishes.

During the second phase of analysis, ALTO will try to generalize any attributes: if all children of some parent have the same value for some attribute, the attribute should perhaps be associated with the higher-level object. Thus, ALTO first ask the user whether the attribute value pair `wheels` and 4 should be associated

with **bmw** directly, rather than with **bmw3** and **bmw5**. The user answers affirmatively. ALTO now continues the analysis and discovers that all children of **car** now have the **wheels** attribute with value **4**. Again, the user is asked whether he wants to raise this attribute, and again the user answers affirmatively. The new inheritance hierarchy is shown in figure 3. As the reader can see, in the smaller window on the right, ALTO not displays the various attributes and values for those attributes, but also from which higher object a particular value-attribute pair has been inherited.

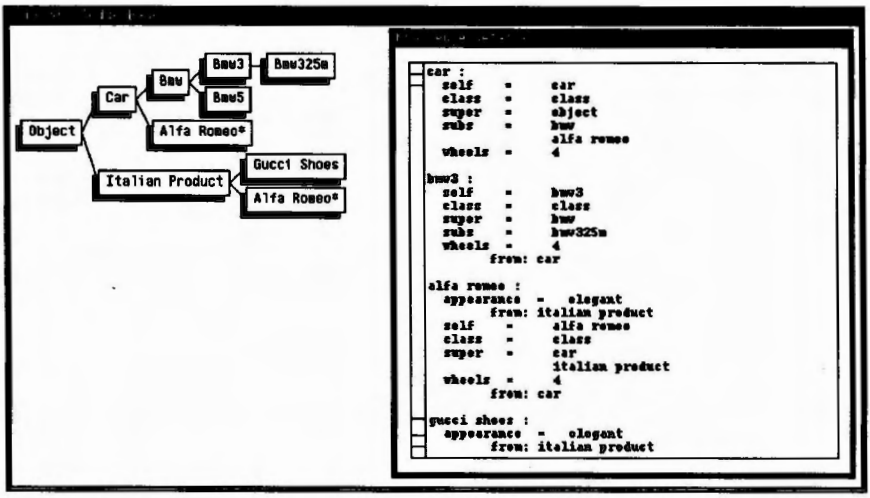



Figure 3: The hierarchy in the car domain after knowledge analysis

The user is still not entirely happy with his hierarchy, and decides to ladder up from **bmw**. Asked for a new parent by ALTO, he replies **german product**. Since this is a new object, ALTO now tries to place this in the existing concept hierarchy. ALTO has basically two possibilities: **german product** can either be a subobject of **car**, or of the top-object **object**. The user answers that **german product** should be a subclass of **object**. ALTO establishes that this new object has the property **durability** with value **good**. It then tries to ladder up from **german product** to establish whether this object is a direct child of **object**. The user says that it is. The final object hierarchy therefore is as in figure 4.

# 5 Implications for a larger KEW

In the previous sections, we briefly mentioned a number of topics that arose in the context of ALTO that may have wider implications for the design of a Knowledge

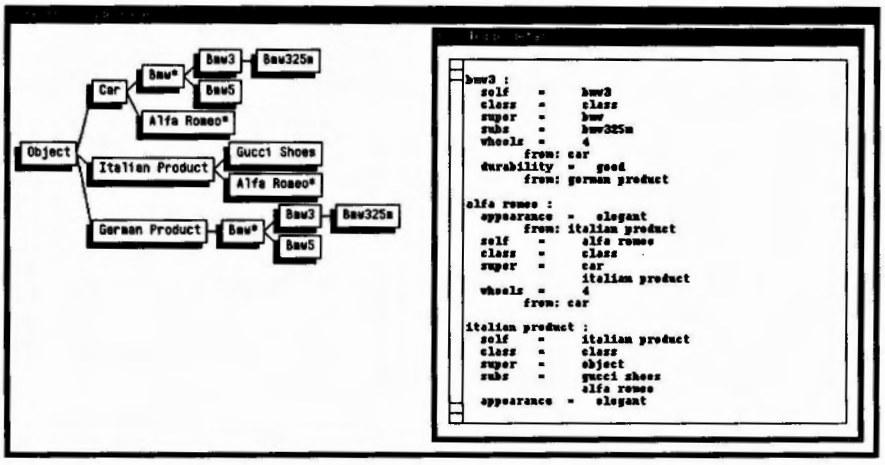



Figure 4: The final hierarchy in the car domain

Engineering Workbench for knowledge acquisition. In this section, we further discuss some of these.

A first set of issues arise from the fact that ALTO supports different settings, which change its behaviour considerably. This points to questions about the rôle of KEW in the knowledge acquisition process. Inexperienced knowledge engineers are likely to need KEW to take the initiative during the acquisition, suggesting both what tools to use in particular situations and how to use them. However, as knowledge engineers become more experienced, they are likely to want to make their own decisions about these issues. This points to the need for KEW to take into account who is using the system and how experienced they are when it is deciding what it should do next. The degree of control in ALTO is determined simply by asking the user whether he or she is experienced, and choosing the appropriate setting. It is an open question to what extent this is sufficient to obtain the kind of flexibility that would seem to be required in a larger KEW. One could imagine more sophisticated means of deciding which group a user was in and changing between them. For example, one could keep records of individual users over many sessions with the system. KEW could then use this information to build up a user model, which could then be used not only to determine the appropriate level of system initiative in different tools, but also in deciding the amount and content of KEW's advice and guidance to a user.

Another set of issues that is illustrated by ALTO follows from the choice of CommonSloop as its knowledge representation language. ALTO is intended to be integrated into a larger knowledge engineering workbench that incorporates a number of different knowledge acquisition tools. In order to make KEW more than

simply a set of tools, it will be essential that the different tools can cooperate, and that knowledge produced by one tool can be integrated with the knowledge produced by another. Since different tools are likely to use different formalisms for representing knowledge, it follows that integration will in general first involve a process of transformation. Clearly, the process of transformation should preserve the meaning of the knowledge represented in the input representation. A transformation that makes true knowledge false, or vice versa, would not be satisfactory. However, this in turn seems to imply that we have a way of ascribing meanings to knowledge bases independent of the way in which this knowledge is represented. In other words, we need to have an interpreter-independent way of ascribing meanings to pieces of knowledge. The only way in which one can do so is by defining a model-theoretic semantics for a knowledge representation language.

Various attempts has been made to define a model-theoretic semantics for frame-based languages (e.g., Attardi and Simi, 1982; Reichgelt, 1985, ch 5; Brachman *et al.*, 1987) However, none of these languages supports default inheritance. Nado and Fikes (1987) attempt to provide a model-theoretic semantics for a frame-based language that does support default inheritance. However, they do so by translating it into default logic (Reiter, 1980), and the problem of providing a model-theoretic semantics for default logic remains open. Brachman (1985) discusses the problem of default inheritance in more detail.

The question that arises from these considerations is to what extent the use of an object-oriented language which supports default inheritance is possible in a larger knowledge engineering workbench. If transformation and integration of knowledge require that one be able to define a model-theoretic semantics for the knowledge representation language used by a tool, and if it is indeed not possible to define a model-theoretic semantics for such a language, then the use of such a language in KEW would seem seriously flawed. Obviously, similar arguments would apply to any knowledge representation language for which there is no interpreter-independent semantics. In the context of ALTO, one way out of this dilemma would be to change the inheritance mechanism used by CommonSloop from inheriting information by default with the possibility of overriding inherited values, to always inheriting and having no override.

# 6 Conclusion

The automation of one knowledge elicitation technique immediately suggests the need to automate more of the process and ultimately all the development cycle of a knowledge-based system. Currently we are working on a prototype KEW which will integrate a number of different knowledge elicitation and machine learning tools in a common environment. The idea is for these tools to work together to form a consistent, complete and integrated knowledge base. A further aim of this prototype is to investigate the extent to which the knowledge engineering process can be au-

tomated, or at least supported, by knowledge-based reasoning about the knowledge acquisition process.

# 7 References


Attardi, G. & Simi, M. (1982) Semantics of inheritance and attributions in the description system Omega. *MIT A.I. Memo 642.*

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

Brachman, R. (1985) "I lied about the trees" Or, defaults and definitions in knowledge representation. *AI Magazine, 5(3), 80-93.*

Brachman, R., Gilbert, V., & Levesque, H. (1985) An essential hybrid reasoning system: Knowledge and symbol level accounts of KRYPTON. *IJCAI 9*, 532-39.

Burton, A.M., Shadbolt, N.R., Rugg, G., & Hedgecock, A. (1988) Knowledge elicitation techniques in classification domains. *ECAI-88*, 85-90.

Clancey, W. (1983) The epistemology of rule-based expert systems: A framework for explanation. *Artificial Intelligence*, 20, 215-51.

Clarke, B. (1987) Knowledge Acquisition for Real Time Knowledge Based Systems. In *Proceedings of the first European Workshop on Knowledge Acquisition for Knowledge Based Systems.*

Gammack, J.G. (1987a) Formalising Implicit Domain Structure. *Proceedings of the SERC workshop on knowledge acquisition for engineering applications.* SERC report RAL-87-055.

Gammack, J.G. (1987b) Different Techniques and Different Aspects of Declarative Knowledge. In Kidd, A.L. (Ed.) *Knowledge Acquisition for Expert Systems: A Practical Handbook.* New York: Plenum Press.

Grover, M.D. (1983) A Pragmatic Knowledge Acquisition Methodology. *IJCAI-83*, 436-38.

Hart, A. (1986) *Knowledge Elicitation for Expert Systems.* London: Kogan Page.

Hinkle, D.N. (1965) The Change of Personal Constructs from the Viewpoint of a Theory of Implications. *Ph.D thesis.* University of Ohio.

Hoffman, R.R. (1987) The Problem of Extracting the Knowledge of Experts from the Perspective of Experimental Psychology. *AI Magazine*, 8.

Johnson, P.E., Zualkernan, I. & Garber, S. (1987) Specification of Expertise. *International Journal of Man-Machine Studies*, 26.

Marty, J-C., Shadbolt, N. R., & Wielinga, B. (1989) The ACKnowledge Project: Improving the Knowledge Acquisition Process. *Results and Progress of Esprit Projects in 1989, XIII/416/89.*

Nado, R., & Fikes, R. (1987) Semantically sound inheritance for a formally defined frame language with defaults. *AAAI 87*, 443-449.

Reichgelt, H. (1985) Reference and quantification in the cognitive view of language. *Ph.D Thesis. Centre for Cognitive Science.* University of Edinburgh.

Reichgelt, H. (1987) Manual for CommonSloop. *Artificial Intelligence Group, Technical Report, Oct, 1.* University of Nottingham.

Reiter, R. (1980) A logic for default reasoning. *Artificial Intelligence*, 13, 81-132. (Also in M. Ginsberg (ed.) (1987) *Readings in nonmonotonic reasoning.* Los Altos, CA: Morgan Kaufman.)

Rugg, G., McGeorge, P. & Shadbolt, N. (1990) On the use of laddered grids in knowledge elicitation. *Artificial Intelligence Group, Technical Report, Feb, 1.* University of Nottingham.

Schweikert, R., Burton, M., Taylor, N., Corlett, E., Shadbolt, N., & Hedgecock, A. (1987) Comparing knowledge elicitation techniques: A case study. *Artificial Intelligence Review*, 1, 245-53.

Shaw, M.L.G. & Gaines, B.R. (1987) An Interactive Knowledge Elicitation Technique using Personal Construct Technology. In Kidd, A.L. (Ed.) *Knowledge Acquisition for Expert Systems: A Practical Handbook.* New York: Plenum Press.

Weiss, S. & Kulikowski, C. (1984) *A Practical Guide to Designing Expert Systems.* Towata, NJ: Rowman and Allanheld.

# A flexible six-step program for defining and handling bias in knowledge elicitation

Mary A. Meyer, Jane M. Booker, Jeffrey M. Bradshaw*
MS F600, Statistics Group, Los Alamos National Laboratory
Los Alamos, New Mexico 87545, USA

*Advanced Technology Center, Boeing Computer Services
P.O. Box 24346, M/S 7L-64, Seattle, Washington 98124 USA

While concern with the quality of expert knowledge is sensitizing knowledge engineers to the issue of bias, little guidance yet exists on how to construe, recognize, and treat its occurrence in knowledge acquisition. Bias is regarded here as a skewing of the data from some reference point. If the reference point is mathematical or logical rules, then biased data violates these rules, such as occurs with cognitive bias. If the reference point is the experts' expressions of their thinking, biased data are distortions of these expressions, such as resulting from motivational biases. In this paper, we provide information on how to select a definition of bias (cognitive or motivational) and how to choose a combination of steps for handling bias in a particular application. The steps are (1) anticipate which biases are likely to occur in the planned elicitation; (2) redesign the elicitation to make it less prone to these biases; (3) make the experts aware of the potential intrusion of these biases and familiarize them with the elicitation procedures; (4) monitor the elicitation for the occurrence of biases; (5) adjust, in real time, to counter their occurrence; and (6) analyze the data to determine if they occurred. The program includes detailed descriptions of how to perform each of these steps using selected biases as examples.

## 1. Introduction

The presence of bias in expert judgment and in knowledge bases has recently been recognized, but few guidelines yet have been developed for handling its occurrence. Bias in knowledge acquisition is of concern because it can affect the quality and credibility of the knowledge. By its definition, biased data is viewed as undesirable, as skewed in comparison to unbiased data. Thus the presence of bias can cause discrepancies that confuse and slow knowledge acquisition. Bias can also enter into the knowledge base and impair the advice given by the knowledge-based system.

Programs are needed to help knowledge engineers deal with the potential problems posed by bias. By programs, we mean a set of steps that the knowledge engineers can perform to achieve certain ends with respect to bias. Such programs would include information on when bias was likely to be of concern and when particular steps would be beneficial. We envision such programs as serving several purposes:

1. bringing the possible occurrence of bias to knowledge engineers' attention and stimulating them to consciously consider whether they need to be concerned about bias in their applications.
2. causing them to consciously examine what they consider desirable in the data and to explicitly set, rather than assume, the reference point from which bias will be defined. This reference point, be it mathematical/logical rules or the expert's unaltered thinking, can then serve as a base for selecting the elicitation methods (e.g., those which are less likely to create the bias) and the later means for validating or verifying the knowledge base.
3. training knowledge engineers in bias and hopefully lessening their tendencies to introduce bias in their elicitation, representation, and analyses of the knowledge.
4. offering a series of steps on what to do about bias. The knowledge engineers would be guided to follow different steps depending on whether they wished to simply become aware of bias, counter its occurrence, or analyze its effects.

In this paper, expert knowledge will include elicited knowledge, such as the experts' solutions to problems and their descriptions of their mental processes (definitions, assumptions, and algorithms) in arriving at solutions. In addition, knowledge will refer to information that the experts have given in numeric form, such as probability estimates, and nonnumeric form, such as their descriptions of their assumptions. We view expert knowledge as changing as the experts receive new information [1] and even as the experts are engaged in describing what they know [2]. Gaines [3] has provided a foundation for understanding how experts' knowledge and performance evolve through positive feedback processes that work at both the individual and societal level. He proposes that expertise is continuously developed through the experts' exposure to interesting problems for solution and to the data of others in their professional community. Random differences in early performance may be amplified as the most interesting problems are given to those with the strongest records of previous successes. Invitations to scientific congresses and awards of scholarships may further magnify these effects. The role of experts in society can thus be seen as trading their advice for access to problem solving and, hence, learning opportunities.

The evolving nature of expertise has implications for knowledge acquisition. The knowledge acquisition process must reflect the changing nature of knowledge and allow for continual updating of the knowledge base or system [3]. Knowledge acquisition occurs in phases where the knowledge base is repeatedly refined and where different kinds of validation are performed at the different phases [4]. The approach to handling bias being proposed in this paper is envisioned as fitting into this dynamic environment of evolving knowledge bases or knowledge-based systems. Particularly, the approach to bias could be used during any phase of knowledge acquisition when expert knowledge is being elicited. Then too, it could be used as part of the validation process proposed by Benbasat and Dhaliwal [4], particularly when the validation check calls for comparing the knowledge in the knowledge base or system to that of the expert source.

### 1.1. The Meaning of Bias

Bias cannot be understood without a context because it is identified as *bias* in comparison to some reference point. Bias is generally defined as a *skewing*, so its definition includes the concept of its counterpart--that which is considered *unskewed* or *not biased*. Implicit in this general definition of bias is the view that what is *not* biased is *reality* or *truth*. For this reason, a brief discussion of the nature of reality and of human perception is necessary to clarify the meaning of *bias*.

In this paper, we hold the assumption of *constructive alternativism* [5]. According to this view, *reality* does not reveal itself to us directly, but rather is subject to as many different constructions as we are able to invent [6,7,8]. How we perceive reality, then, is through the filter of our mental models. Bateson [9] gives the example of someone stepping on his toe. He argues that what he experiences is not the stepping on of his toe but the image of this event from neural reports reaching his brain after the event has happened. He concludes that there is always "a transformation, a coding between the report, and the thing reported...." Not only the quality of our emotions but also the actual degree to which we experience pain is determined by our attributions about the context of why, how, and by whom the act was performed. Our memory for the event, our perceptions of the present, and our anticipations of the future can never be *objective* because they arise from an inevitably subjective and personal model of events [10].* As Shaw and McIntyre [11] observe "Ideas are not in the mind, nor objects in the world, but... both are in the meeting of mind and matter."

Similarly, we cannot decide objectively what is biased and what is not. We can only make value judgments as to what we consider desirable (e.g., more objective) in the data. Langer, in her book *Mindfullness* [12], demonstrates that all our decision making is ultimately based on individual values [10]. To illustrate that decisions are not made solely on the basis of data, as we typically believe, she takes the example of those who must decide whether to prolong life when a patient is in intolerable pain. Receiving additional data does not enable the persons to make a decision because no amount of information will make one answer absolutely right. Then too, there is no

*The fact that we usually take our models of reality for reality does not invalidate the above argument. As Bateson has stated (1979, p. 31) "Our civilization is deeply based in this illusion." At the cultural level, meaning and structure are imposed and then taken for reality by members of that culture. For example, members of the Western scientific culture would take the color spectrum, such as in a rainbow, and divide it into four to six colors--violet, blue, green, yellow, orange, and red. In another culture, the people would not see this segmentation of colors. Instead, they may have been conditioned to view the spectrum as consisting of wet and dry colors. The members of both of these cultures have been conditioned to see color in a particular way and to believe that it objectively exists as they perceive it.

logical stopping point to cue the decision makers on when to end gathering data. Eventually, the discussion of whether to prolong life returns to the basis of individual values. "The doctor, judge, and patient must decide between the principle of prolonging life at all costs and the 'right' to determine the quality of life."

Because our decision making is based on value judgments, it is appropriate that the term *bias* have a negative connotation. Bias is what we label as undesirable or less good. Our view of bias can be likened to dirt in Douglas's anthropological portrayal of ritual pollution. "As we know it, dirt is essentially disorder. There is no such thing as absolute dirt: it exists in the eye of the beholder...Eliminating it is not a negative movement, but a positive effort to organise the environment.... In chasing dirt, in papering, decorating, tidying we are not governed by anxiety but are ...positively re-ordering our environment, making it conform to an idea." [13].

If we accept that defining bias is a value judgment, then we have the option of explicitly articulating presuppositions about the view we hold, so that the presuppositions themselves can be criticized and improved [9,14]. One way in which these suppositions can be improved is by testing them as hypotheses, as described in Feldman, Compton, and Symthe [15]. Thus, the views that we present should be seen not as a *depiction of what bias is* but rather as a *vehicle for the attainment or formulation of greater knowledge about bias* [16]. The critical question is not *How do we know that our hypotheses about bias are correct?* (every hypothesis being an incorrect simplification) but *How can we expose our hypotheses to maximum criticism, in order to evaluate and refine them as thoroughly as possible?* [14].

From a constructivist point of view, there are two criteria for evaluating the adequacy of a definition of bias: (1) its coherence within a system of related ideas, and (2) its usefulness in predicting and controlling the phenomena in question. By coherence is meant "the regularity that binds the ideas together" [9]. For instance, the fields which have contributed approaches to knowledge acquisition contain their own implicit and internally coherent beliefs as to what good data is, what bias is, and what methods can be used to achieve better data. Knowledge engineers can examine whether the definition of bias which they are considering would be consistent with these ideas. A definition of bias can also be evaluated according to "what is the most useful way of construing the problem from the point of future prediction and control" [17]. For example, if a project's aim was the validation of the expert's estimates against logic, a view of bias that defined bias as a violation of logical rules would be most useful.

## 1.2. TWO VIEWS OF BIAS AND THEIR PLACES WITHIN KNOWLEDGE ACQUISITION

Artificial intelligence, decision analysis, and social science literature contain two somewhat different views of bias. These two views differ in terms of the reference point from which bias is judged. The reference point for one definition of bias is what the source expert actually knows or does; the reference point for the other definition is Western science's concept of *objective* criteria for reality. In the descriptions below we have exaggerated the distinctions between the definitions for the purpose of illustration.

Those who are looking from the first reference point have concerned themselves largely with kinds of biases that we will term *motivational*. For example, the conscious or unconscious accommodations that an expert makes to please the knowledge engineer constitute forms of motivational bias. The concern with motivational bias is prevalent in the soft sciences, particularly psychology and cultural anthropology. Practitioners of these disciplines generally consider the expert's way of thinking to be the *gold standard* that they wish to capture through the elicitation or the building of a knowledge-based system [18].

With the second reference point, bias is defined as occurring when the expert's knowledge does not follow *objective* rules or standards considered to apply to the situation. We group these deviations under the heading of *cognitive bias*. Thus, if the expert's predictions do not match what can be observed and measured, cognitive bias will have occurred according to this definition. However, expert's knowledge often cannot be checked against accepted facts or physical measures of the phenomena. Thus, cognitive bias is frequently determined by checking the expert's data against the mathematical standards that Western scientists apply to testing information. To illustrate, if an expert gives probability estimates on all outcomes to a problem (previously defined as being mutually exclusive) and these probabilites do not sum to 1.0, these data will be considered biased because they do not follow normative statistical rules. This view of bias comes from the fields of decision analysis [19] and statistics. Decision analysts do not deny the importance of research in describing how people actually go about making decisions but insist that research be applied to discover ways to counteract systematic sources of bias in intuitive judgment [20, 21].

These views of bias are often implicit in knowledge acquisition applications or, at least, in particular phases of the application. We have observed that these views coincide with particular project goals, types of elicitation techniques, selections of expert response modes, and validation strategies.

The motivational view of bias, for instance, is more prevalent in knowledge acquisition projects whose goal is the psychological modeling of the source expert. In addition, proponents of this view of bias are likely to favor the use of nondirective elicitation techniques in the desire to avoid *leading* the expert. The verbal protocol technique, in which the expert thinks aloud but is not questioned by the knowledge engineer, is one such nondirective method. Also proponents of this view tend to avoid complex numeric response modes and allow the experts to respond spontaneously in their own, often nonnumeric, terms. Additionally, the motivational view of bias can be found in validations that focus on how well the knowledge base or the the knowledge-based system corresponds to the source expert's knowledge.

An emphasis on the cognitive view of bias is likely to coincide with projects whose goal is to improve on the expert's judgment. In projects where the concern with cognitive bias dominates knowledge acquisition, the expert may be asked to respond using forms that allow greater precision of expression. For instance, it is easier to determine that experts are underestimating uncertainty if they are using probabilities than if they are simply stating that some event is very likely. Thus, the formal checks on the expert's knowledge can be done from the very first phases of knowledge acquisition. The focus on validation from the cognitive bias perspective can also occur in later validation phases. For example, Gaines [3] describes one validation criteria of a knowledge-based system as checking *to what extent the knowledge in the system corresponds to actuality.*

## 1.3. SOURCES OF BIAS

Different reasons are advanced for the presence of each kind of bias. Motivational bias is assumed to be driven by our human needs, such as for approval, whereas cognitive bias is generally seen as being driven by constraints on our information processing.

### *1.3.1. Motivational bias*

We present motivational bias as occurring when: (1) the experts do not report what their solutions or thought processes actually were, (2) the knowledge engineer misinterprets the experts' reports, or (3) the knowledge engineer misrepresents the experts' knowledge.

In the first situation, the altering of the experts' reporting, the experts may change their descriptions of their thinking because of concerns over the social desirability of their responses. For example, if the knowledge engineer asks the experts a leading question, such as *Did you use subgoal x in solving the problem?*, they may feel they should answer "yes," even if they did not solve the problem in this way. The mode in which the experts are asked to respond can also bias the experts' answer if they cannot accurately encode their final judgment into it. Some common numeric response modes are probability distributions, continuous linear scales, Saaty's paired comparisons, ratings, and rankings. Then too, the knowledge engineer's verbal or nonverbal responses can influence the expert's reporting. For instance, if the knowledge engineer leans forward, displaying intense interest in something that the expert is saying, the expert may unconsciously respond by giving undue emphasis to statements on this topic, and ignoring relevant data that evoked a lukewarm response. The reactions of other experts present in the room can have similar effects on what the expert says. Furthermore, experts' thinking can be affected by their imagination of how those not physically present, such as clients or supervisors, might view their responses. For example, in a study examining whether magnetic fusion projects would meet their technical milestones, some of the experts admitted that they considered their organization's judgments in giving their estimates [22].

There are several reasons why experts' reporting of their thoughts can be influenced by others. First, most people have an emotional need to be accepted and to receive approval [23]. Thus, in elicitation situations, experts are likely to be responsive to what they believe their managers, other experts, or the knowledge engineer wish to hear. Secondly, people are generally unaware of how they make decisions, such as in solving problems [24]. Yet, it is difficult for them to admit ignorance because Western scientific tradition assumes that the problem-solving process can be precisely stated [25]. Experts are likely to fabricate or to acquiesce to suggestions of an acceptable answer or to means by which they might have solved the problem.

The expert's verbalizations can be biased by another source--the knowledge engineer's interpretation of the expert's words. We, as humans, tend to perceive and interpret incoming information in a selective way that supports what we already believe. Sometimes, this leads to a

misinterpretation, or biasing, of the information. For instance, when the expert mentions a new term, we tacitly assume a meaning based on our experience with similar sounding words. Likewise, we may think that we hear the solution that we expect. Knowledge engineers can be particularly prone to this bias because of a method that they have used for learning the expert's field. This method is for knowledge engineers to study the expert's domain, to build mental constructs representing the knowledge, and to understand the expert's knowledge structure by comparing it to their own. This comparative means of learning the expert's knowledge structure would seem prone to bias because the knowledge engineers interpret the expert's thoughts through the filter of their own.

The elicited knowledge can also be altered by the representation of it. Expert data are often modeled for analysis, such as for detecting bias or for building knowledge-based systems. The individuals performing these tasks makes many tacit decisions and assumptions about the data's appearance and performance. For example, when a model is selected for the data, the model assumes particular properties of the data, such as its distribution. These assumptions may not be valid. Then too, the knowledge engineers may have to aggregate multiple and differing expert judgments to provide one input value. Mathematical aggregation schemes often require assumptions, and different schemes, like the mean versus the weighted average, can produce different answers. The knowledge engineers also make decisions about the organization of the knowledge in the system and its representation that determine how the data are implemented. For example, the knowledge engineers may wish to use particular system building tools that lead to "wrong, inefficient, or opaque structures being imposed on the data." [26].

### *1.3.2. Cognitive bias*

The human mind is limited in how much information that it can process and remember [24]. In order to reduce the cognitive burden, people tend to take short cuts when solving a complex problem. Thus, they start with a first impression and integrate the information in a sequential manner, making only a few minor adjustments. Later, if additional information is received, they probably will not adjust their initial impression to give a more accurate judgment. In other words, if individuals who have already reached initial solutions are given contradictory data, they will probably not take this data sufficiently into account when generating final answers. In particular, this sequential means of integrating information handicaps us in making predictions where large or sudden changes are involved. This limiting effect is called *anchoring* or *anchoring bias*.

The human mind has limited memory capacity for information processing. As Miller [27] has noted, most individuals cannot discriminate between more than seven things at a time. This limitation in information processing causes people to be inconsistent in working through the problem. For instance, people commonly forget an assumption made earlier and contradict it, thus becoming inconsistent in their reasoning. This is one example of *inconsistency bias*.

Then too, some data is more easily recalled than others. For instance, data involving catastrophic, familiar, concrete, or recent events may be easier to recall [28, 29]. This effect, termed *availability bias*, can lead to the overestimation of the frequency of some events.

## 1.4. IMPORTANCE OF BIAS TO KNOWLEDGE ACQUISITION

That the problem of bias will be important to knowledge acquisition follows from our definition of bias. In this paper, we have argued that bias is data that is judged to be undesirable, whether this data is evaluated in terms of matching the expert's thoughts or *objective* standards. Thus, the inclusion of biased data in the knowledge base is to degrade the perceived quality of this knowledge. For example, if motivational bias occurs in the form of the knowledge engineer misinterpreting and then misrepresenting the expert's knowledge, the elicited knowledge will not match that of the expert source at the time of the elicitation. If the expert's knowledge is being used as the reference point for validating this information, the information would be judged invalid. The same problem occurs with cognitive bias. For example, a plant pathologist gave a higher expected value to fungicide treatment than the sensitivity analysis performed on the knowledge base yielded [18]. The analysis showed that the fungicide treatment had a lower value than the expert judged because the infection is difficult to cure and the fungicide not highly effective. If this cognitive bias was not checked, the implemented knowledge-base system could have provided advice which was not mathematically sound.

In addition, bias needs to be treated because of its negative impact on others' perception of expert knowledge and knowledge-based systems. Several articles have recently questioned the accuracy of expert clinical judgments in psychology and psychiatry [30, 31]. The credibility of

expert knowledge is open to this same sort of questioning if no attempts have been made to examine and, if necessary, counter bias.

Section 2 of this paper describes a six-step program for handling bias. The first part of the program leads the reader through the process of selecting a combination of steps for his or her application. Guidance is given, then, on how to follow each step using examples from both motivational and cognitive bias. A summary of the work is given in Section 3.

## 2. Program for handling bias

Approaches to handling biases are rare [28] and in their early stages. They are perhaps as much art as they are science. The program proposed here is no exception. One reason that there are not more programs for handling bias is that it is a difficult topic to study. Studying, much less trying to counter bias, is complicated by not having a readily available baseline from which to determine the direction and magnitude of the bias. For questions where expert knowledge is elicited, there are frequently no known *right* answers or data. Thus, the expert's data cannot be simply compared to a known value or to the result of a model. Measurement of motivational bias is further complicated by the absence of *objective* standards for comparison. At least with cognitive bias there are traits of the expert's answers that can be compared to standards. For example, do the expert's mutually exclusive and exhaustive probabilities sum to one as they should? With motivational bias, the baseline of the expert's knowledge is difficult to determine, especially because expertise is not static. In his book on memory, Rosenfield [32] argues that higher mental functions are not fixed procedures but subject to constant reconstruction. While we recognize that both biases are difficult to detect, we believe that for progress to occur, programs like this one must be proposed and applied to knowledge acquisition.

Our approach differs from the one presented by Cleaves [28]. First, Cleaves' proposal focuses on cognitive bias; second, Cleaves tries to anticipate biases by the judgment processes in which they are likely to occur; namely hypothesis and solution generation, decision rule articulation, uncertainty assessment, and hypothesis evaluation. While we agree that biases occur during these processes, we try to anticipate the biases by the elicitation components that are likely to exhibit them. We assumed that many knowledge engineers were lay persons in the areas of human cognition and that this approach might be easier for them, at least as a starting point.

Another major difference in our program is its real-time emphasis. Given the evolving nature of expertise, bias is best detected when it is being elicited. This program stresses monitoring and adjusting for bias, particularly motivational bias, in real time rather than mathematically compensating for it afterwards. In particular, it is much more difficult to determine that motivational bias has occurred after the elicitation because the baseline--the expert's knowledge--is likely to have changed. For this reason, we consider each elicited datum to be a snapshot that can be compared to a snapshot of the expert's state of knowledge at the time of the elicitation.

Our proposed program consists of these general steps:

1. Anticipate which biases are likely to occur in the planned elicitation.
2. Redesign the planned elicitation to make it less prone to the anticipated biases.
3. Make the experts aware of the potential intrusion of particular biases.
4. Monitor the elicitation for the occurrence of bias.
5. Adjust, in real time, to counter the occurrence of bias.
6. Analyze the data for the occurrence of particular biases.

### 2.1. DETERMINING WHICH STEPS TO APPLY

The steps of the program above can be applied in sequence or singly, depending on the needs of the project. For example, if information on bias were not important to the project, none of the steps or only step 6, analyzing for bias, need be applied. If on the other hand, the knowledge engineers wish to follow some but not all of the steps, they could do so. We suggest that step 6 always be done regardless of the other steps because it provides a general check on the expert data.

To pick steps for use in a project, we suggest that the knowledge engineers consider the following: (1) their reason for addressing the problem of bias; (2) the view of bias, motivational or cognitive, that will be employed; and (3) the forms of bias in which they are especially interested.

#### *2.1.1. The reason for focusing on bias*

The knowledge engineers' reasons for focusing on bias can provide a criteria for determining which steps of the program to implement. For example, if the knowledge engineers' interest in

bias stems from a desire to avoid having reviewers criticize aspects of the project, their selection of steps would probably differ from those whose goal is to analyze the knowledge base for the presence of bias. It is also possible that, on some projects, there would be no reason for delving into the problem of bias. In these instances, we recommend that bias *not* be investigated because this activity demands extra time and resources. Instead, we would suggest that the knowledge engineers record their reasons for not investigating bias and proceed as planned with the project.

We have identified several reasons for focusing on bias: (1) to become aware of bias, (2) to prevent or inhibit the occurrence of bias, (3) to avoid criticism of the quality of the knowledge base, and (4) to analyze the data for the presence of bias. These aims can be accomplished by using different combinations of the six steps.

*2.1.1.1. To become aware of bias.* Wishing to learn about bias is similar to wanting to discover and articulate one's assumptions. In both cases, awareness allows individuals to consciously examine the phenomena and choose their beliefs or actions as opposed to blindly exhibiting them. To become aware of bias, we recommend using step 1, *anticipate bias*, and 3, *make the experts aware of the potential intrusion of particular biases*. If the knowledge engineer only wishes to learn more about biases for a particular application, only step 1 is necessary. If, however, the knowledge engineer also wants the experts to be aware of bias, both steps 1 and 3 will be needed. This is because the knowledge engineer is the one who informs the experts about bias and the elicitation procedures. We consider familiarizing the experts with the elicitation procedures part of the process of informing them about bias. First, the elicitation methods form a backdrop against which bias can occur. Second, acquainting the expert with the elicitation procedure may prevent some of the expert's confusion, a contributor to the occurrence of many biases.

Step 4, *monitor the elicitation for the occurrence of bias*, can also be used to sensitize the knowledge engineer to bias. Occasionally, the experts can assist the knowledge engineer in looking for the signs of selected biases.

*2.1.1.2. To prevent or inhibit the occurrence of bias.* If the reason for focusing on bias is to prevent or inhibit its occurrence, we recommend using at least steps 1, 2, 3, 5 and 6. Step 1 enables the knowledge engineers to anticipate the likely biases; step 2 allows them to use this information to redesign the elicitation in making it less susceptible to these biases. Step 3, *make the expert aware*, can also be implemented to reduce the likelihood of the expert introducing bias. Step 5, *adjust in real time to counter the occurrence of bias*, should be executed very carefully because the countering techniques could add bias or complicate later analyses. Step 6, *analyze the data for the occurrence of particular biases*, is performed as a check. Without step 6, there is only the knowledge engineer's claim that steps 1, 2, 3, and 5 have prevented bias. In our experience, conducting step 6 has usually allowed the knowledge engineers to state that no evidence was found of particular biases.

Although not necessary, step 4, *monitor the elicitation*, can also be performed as part of the goal of preventing the occurrence of bias.

*2.1.1.3. To avoid criticism of the quality of the data.* Sometimes the reason for focusing on bias comes from forces outside the project, such as others' concerns with the quality of the data, rather than from within the knowledge engineer. We have noticed that those who have had the unpleasant experience of having their project's data criticized by outside reviewers are highly motivated to avoid this situation in the future. This motivation is different from the other three mentioned in this section in that those who are motivated by this desire will probably not be interested in performing more than the minimum of steps, in this case, step 6. However, to analyze the data for bias assumes some knowledge of bias, so step 1, *anticipate the biases*, might also be necessary. In addition, step 4, *monitor the elicitation*, can be done to provide data for the analysis.

In our experience, external reviewers have most commonly questioned data regarding the presence of these biases: social pressure from the knowledge engineer, social pressure from group think, and wishful thinking by the expert.

*2.1.1.4. To analyze the data for the presence of bias.* Analyzing the data for the presence of bias may be done for a variety of reasons. The knowledge engineers may be interested in analyzing for the presence of bias as a means to an end, such as to study the causes and effects of bias, or as an end in itself. If the knowledge engineers wish to study bias, particularly the techniques and situations that affect its appearance, any of the steps (1, 2, 3, 4, and 5) could be used in combination with step 6. For example, if the knowledge engineers were investigating the effect of using a particular step or elicitation technique, they could use it in one study and not in the control study, and then analyse and compare the results. One advantage of studying bias in this way would be the fresh look at bias that it would produce. For instance, the researchers could determine if there was a difference in applying the step and whether they judged that difference to be positive or negative. In other words, the researchers would not automatically assume that bias always leads to poorer results but would evaluate for themselves the peculiar properties of bias or its absence. (For this idea, we are indebted to Maurice Sharp, University of Calgary, Canada).

If analyzing the data for bias is the aim, such as for validating a knowledge base, steps 1 and 6 could be used. Step 1 allows the knowledge engineers to learn about likely biases and step 6 enables them to check for the presence of these biases. Generally, step 6 is more effective in investigating sources of cognitive bias. For example, the

knowledge engineers could check for underestimation of uncertainty, as described in section 2.3.5.1. If, however, the knowledge engineers want to analyze the data for how well it matches the expert's thinking (as would be the case with the motivational definition), we would suggest adding step 4. Step 4, *monitor the elicitation*, enables the knowledge engineers to watch for some of the motivational biases that interfere with obtaining reliable reports of the expert's thoughts.

### *2.1.2. The selection of the view of bias, motivational or cognitive*

While the view of bias, motivational or cognitive, is often implicit in a knowledge acquisition project, we recommend that the knowledge engineers consciously select which view or views of bias to apply. By consciously selecting the view of bias, the knowledge engineers can consider which view will best serve the application in terms of coherence and usefulness and which set of steps will be most effective. For example, if the goal of the project is learning the expert's problem solving in order to emulate it, the motivational definition of bias would be more appropriate. If the project involves estimating the likelihood of future events, the cognitive definition would be a natural choice because people are inaccurate in making predictions, and the cognitive view of bias would help to combat this weakness.

We suggest that the reader select and use only one view of bias at a time to avoid being contradictory. For example, use of the cognitive definition would propose that a mathematically incorrect judgment be modified. This act would cause a misrepresentation of the expert's data, a bias, according to the motivational definition of bias. Thus, we recommend that the knowledge engineers' formulate rules for when each view of bias will be applied, if more than one view is to be used. To illustrate, the definition of motivation bias could be used in acquiring knowledge for building the conceptual knowledge base and the cognitive definition used thereafter.

*2.1.2.1. Motivational definition.* As a general rule, if the motivational definition of bias is selected, steps 1, 2, and 3 will be most helpful. In step 1, the knowledge engineers are given information on what biases to expect. In step 2, they refer to the *Definitions* (section 2.2.1.3.) to receive ideas on how to minimize bias in the planning or the redesigning of the elicitation. For example, if looking at the index led the knowledge engineers to expect group think, the definition of group think would provide them with ideas on how to design an elicitation that was less prone to it. Group think is more likely when behavioral aggregation is used so the knowledge engineers could plan to aggregate the responses mathematically. Use of step 3 *make the experts aware of the potential intrusion of bias and familiarize them with the elicitation procedures* would lead the knowledge engineers to inform the experts about bias, and hopefully make the experts less prone to it. In this book, we have focused on presenting methods of elicitation and analysis which we believe minimize influencing the experts and force fitting their data. Thus, just using the methods suggested in this book, regardless of any program for handling bias, should provide some protection from motivational bias. In addition, step 4 *monitor the elicitation* is often helpful in detecting motivational biases such as group think. The *Signs of Selected Biases* (section 2.2.3.1.) tell the knowledge engineer which behaviors might be indicators of particular biases.

*2.1.2.2. Cognitive definition.* If the cognitive definition of bias is selected, step 6 *analysis* is particularly effective. Analysis is generally more definitive with cognitive than motivational sources of bias because cognitive biases can be determined mathematically or statistically. Cognitive bias can usually be measured because it is defined as a violation of logical or statistical standards. Thus, three of the four cognitive biases mentioned in this paper should be included in the analyses. For example, *underestimation of uncertainty* can be analyzed using the experts' ranges on their estimates (as described in 2.2.6.1.). Inconsistency is analyzable through logic checks. Anchoring can be investigated by comparing experts' first impressions to their later ones, given that the problem environment has changed, such as through the receipt of new information. Analysis of cognitive bias presumes that the knowledge engineer knows which biases might pose problems. Thus, step 1 is useful in dealing with cognitive biases. In addition, step 3, *make the expert aware of the potential intrusion of bias and familiarize them with the elicitation procedures*, is often used with cognitive biases to lessen the expert's tendencies to introduce these.

### *2.1.3. Interest in particular types of bias*

Frequently, those working on a project are more interested in some types of bias than others. The knowledge engineers interest in specific biases can be used to select the steps that are more effective with those biases. For the three selected biases below, we describe the appropriate steps.

*2.1.3.1. Social pressure from the knowledge engineers.* For instance, steps 1, 2, and 3 would be most helpful in dealing with social pressure from the knowledge engineers. In doing step 1, the knowledge engineers would become aware of their tendency to introduce this bias and perhaps be able to curb it. As a result of step 2, they might decide

to select elicitation methods that were nondirective, like the verbal protocol, to minimize their tendency to influence the expert. In step 3, the experts would be made aware of this bias so they would be able to guard against it.

*2.1.3.2. Social pressure from group think.* For dealing with social pressure from group think, all the steps may be necessary. In step 1, the knowledge engineers are able to assess whether their application is susceptible to group think. Assuming that it is, they can redesign the elicitation using information from step 2. Step 3 allows the knowledge engineers to alert the experts to the potential occurrence of this bias, and step 4 allows both parties to monitor the sessions for its occurrence. In step 5, the knowledge engineers can apply techniques for countering this bias. In step 6, the data may be analyzed for the presence of group think.

*2.1.3.3. Wishful thinking.* Step 1 informs the knowledge engineers that wishful thinking can occur in many elicitation methods. Step 2, allows the knowledge engineers to redesign the elicitation to select those experts who were less likely to exhibit wishful thinking, those who had less at stake in the judgments. In one study, whose aim was to obtain probabilities of whether project deadlines will be met on time, we focused on interviewing those physically working on the project rather than their managers [22] Additionally, we required that the experts disaggregate the question into its parts, give estimates for each part, and explain their reasoning for each estimate. This practice was to make it more difficult for the interviewees to give highly optimistic estimates.

## 2.2. HOW TO EXECUTE SPECIFIC STEPS

### *2.2.1. Step 1--Anticipate potential biases in the elicitation*

Anticipation of potential biases is the first step in the proposed program. This process is illustrated using nine frequently encountered biases. These were selected because they represent a range of bias within the two definitions. Certainly these are not the only types of bias. However, they are ones that we have frequently encountered and for this reason are used as a starting point.

To anticipate bias, the knowledge engineers determine the parts of their planned elicitation using the *Components of Elicitation* below as an aide. They then search the *Index of Selected Biases* (section 2.2.1.2.) for the biases to which these components are prone. For instance, if the knowledge engineers planned to use the interactive group method, they would see that it was prone to the following biases: social pressure from group think, wishful thinking, and inconsistency. They then turn to the section after the index (*Definitions of Selected Biases*) to look up the definitions and causes of the selected biases. Through the process of looking up these biases in the *Index* and *Definitions*, the knowledge engineers will become aware of the biases' existence and of their own tendencies to introduce or acerbate them.

#### *2.2.1.1. Components of elicitation*

**1. The elicitation situation** is where the knowledge engineer communicates with the expert. There are three distinct situations which can overlap each other in practice.

**The individual meeting** is the situation where the knowledge engineer meets with an expert face to face. The individual meeting could be focused, structured, or unstructured in its questioning and include problem discussion, description, or analysis, as described by Boose and Gaines [33]. Most of the elicitation techniques, even those for eliciting problem-solving data (e.g., protocol analysis, the verbal probe or the ethnographic method), can be used.

**The interactive group meeting** is defined as where the knowledge engineer meets with the expert in the presence of other experts. Usually the knowledge engineer moderates these sessions and leads the questioning. This elicitation situation can range from traditional meetings, to negotiations, to decision-analysis sessions, and to interactions on terminals in a computer network [34]. The interactions of the group members can be structured to differing degrees. An example of an unstructured interactive group would be the traditional business meeting; a highly structured interactive situation would be one in which the experts were requested to provide their responses to each other in specific forms at particular times. For instance, experts in army export control were required to discuss the problem and then give their data in the form of estimates with a few sentences explaining their reasoning [35]. Elicitation techniques that provide broad coverage (e.g., the verbal probe), as opposed to in-depth focus, are best used in the interactive group situation.

**The Delphi** situation occurs when the knowledge engineer keeps the experts physically separate but acts as a channel for distributing each expert's information to all of the experts. (This technique was developed by Rand Corporation to avoid having members of the group influence each other's responses.) In strict Delphi, the knowledge engineer obtains the experts' responses through the mail, makes these responses anonymous, redistributes these anonymous responses to the individual experts, and allows the individual experts to revise their previous responses. This process can be repeated as many times as desired. In general, the Delphi has not been used to elicit in-depth problem-solving data because this type of data is best elicited when the knowledge engineer and expert are face to face. Instead, structured survey types of questions are asked in the Delphi situation. The Delphi has been used for classifying problems and for group negotiations [33].

**2. The modes of communication** are the means--face to face, telephone, or mail (electronic or postal)--by which the knowledge engineer and expert communicate during the above-mentioned elicitation situations. Several modes can be combined in one elicitation process. For example, the knowledge engineer can send the expert the problem, answer the expert's questions about it over the telephone, and then elicit the expert's problem solving in person. In addition, modes can sometimes be substituted for one another. For example, in the Delphi situation, the information could, if it were simple and brief, be communicated by telephone rather than by mail.

**3. The elicitation techniques** are the means by which the experts are led to verbalize aspects of their knowledge, such as their definitions for terms, and the processes and procedures by which they solve problems. (The experts' verbalizations of their answers are covered under a separate component--the response mode.) One elicitation technique is interviewing, which itself is composed of different ways of asking questions, such as the verbal probe and the ethnographic technique [36]. Another elicitation technique is the verbal protocol in which the knowledge engineer observes the experts, usually individually, and listens to them as they solve problems. Repertory grids are another elicitation technique whereby the key elements of the expert's thinking are elicited and then related to the distinctions that the expert makes regarding these elements.

In general, the elicitation techniques vary according to: the part of the knowledge acquisition cycle in which they are used (e.g., to extract the structure of the domain or more detailed production rules); whether they are applied while the expert is solving the problem or afterwards; whether they include a whole sequence of techniques to be applied at different points in the acquisition, such as KADS [37] or whether they are only one technique; the degree to which they are automated (e.g., the manual versus automated elicitation of repertory grids), and the amount of structuring (e.g., spontaneous questioning versus interviewing by topic area). The elicitation techniques are not listed in the index below because some, like general interviewing, can be performed in such varying ways that they would be subject to different biases. For a more comprehensive description of elicitation techniques and of the larger set of methods used for knowledge acquisition, see [38, 39].

**4. The response mode** is the form in which the experts are asked to give their answers or solutions to problems. Some times, in knowledge acquisition the experts are allowed to respond naturally rather than encode their thoughts into a form selected by the knowledge engineer. Other times, a numeric response mode, such as probabilities, continuous linear scales, Saaty's paired comparisons, ratings, and uncertainty measures is requested.

**5. The aggregation scheme** is the means by which responses from many experts are combined into one response, such as one line of reasoning or a numerical solution. Aggregation can be mathematical or behavioral.

**Mathematical aggregation** is the use of mathematical means to combine multiple expert's quantitative answers into one answer. Some mathematical methods weight the expert's answers equally (e.g., the mean), others weight the experts' data differently in attempts to give higher weights to the *more expert* or less biased data. Mathematical aggregation is outside the scope of this paper. (For information, see chapter 16 of [40].)

**Behavioral aggregation** relies on the experts reaching a consensus [41]. The aggregation occurs during, rather than after, the elicitation session. When behavioral aggregation is used in interactive group situations, the experts are informed that they must reach a consensus and often employ persuasion and compromise to do so. When behavioral aggregation is used with the Delphi technique, the experts give their responses and receive feedback until they reach one response.

*2.2.1.2. Index of selected biases*

| Components | Source of bias--Type |
|---|---|
| **Elicitation situations** | |
| Individual meeting | Motivational--Social pressure from KE |
| | Motivational--Wishful thinking |
| | Cognitive--Inconsistency |
| Delphi | Cognitive-Inconsistency |
| | Motivational--Wishful thinking |
| | Cognitive--Anchoring |
| Interactive group meeting | Motivational--Social pressure from group think |
| | Motivational--Wishful thinking |
| | Cognitive--Inconsistency |
| **Modes of communication** | |
| Face-to-face | Motivational--Social pressure from KE |
| | Motivational--Wishful thinking |
| | Cognitive--Underestimation of uncertainty |
| Telephone | Motivational--Social pressure from KE |
| | Motivational--Wishful thinking |
| | Cognitive--Availability |
| | Cognitive--Anchoring |
| | Cognitive--Underestimation of uncertainty |
| Mail | Motivational--Social pressure, impression management |
| | Motivational--Misinterpretation by KE |
| | Motivational--Wishful thinking |

| | |
|---|---|
| | Cognitive--Inconsistency |
| | Cognitive--Availability |
| | Cognitive--Anchoring |
| | Cognitive--Underestimation of uncertainty |
| **Response modes** | |
| Complex, numeric response modes, such as probability, distributions, especially if expert inexperienced in use | Motivational--Misinterpretation by expert<br>Motivational--Wishful thinking<br>Cognitive--Inconsistency<br>Cognitive--Underestimation of uncertainty |
| **Aggregation** | |
| Behavioral aggregation | Motivational--Social pressure, group think |

*2.2.1.3. Definitions of selected biases*

**Motivational Bias** is the altering of the experts' responses through social pressure (from the knowledge engineer, group think, or impression management) and misinterpretation. In addition, the experts' motivation, such as in wishful thinking, can influence their responses. Wishful thinking bias differs from the other selected motivational biases in that the experts' expressions are not necessarily changed by the experts themselves or by the knowledge engineer. Wishful thinking is included in the motivational biases because it originates from the expert's motivation.

*Social pressure* is the altering of the experts' descriptions of their thoughts arising from the desire to be accepted and to see themselves in the most positive light possible. This altering can take place consciously or unconsciously. The social pressure can come from those physically present, such as the knowledge engineer or the other experts, or from the expert's own internal evaluation of others' reactions.

**Social pressure from the knowledge engineers** is most likely to occur in those methods where the knowledge engineers are meeting in a face-to-face situation with the experts, such as in the individual and the interactive group meeting. In face-to-face situations, the knowledge engineers can intentionally or unintentionally influence the expert through body language, facial expression, intonations, and choice of words. We expect this source of bias to be weaker in telephone conversations and weaker still in communications by mail because these modes do not allow some of the means of expression in the face-to-face mode. In addition, social pressure bias is more pronounced when the knowledge engineer is asking leading questions. Thus, it is weaker when the knowledge engineer is using one of these elicitation techniques--the verbal protocol, the verbal probe, or ethnographic phrasing of questions [36]. The verbal protocol mentioned here requires the experts to think aloud as they solve a problem and thus avoids leading the experts because it does not involve questioning them; the verbal probe uses general, nonleading phrases; and the ethnographic technique uses the expert's own words in formulating questions. We would also expect other knowledge acquisition techniques, such as KADS [37] and SORTAL [42], that are driven by the expert's terms and meaning to be less prone to social pressure from the knowledge engineer.

**Group think** is another form of social pressure from members in the group inducing individuals to slant their responses or to silently acquiesce to believe what will be acceptable to the group [43]. Zimbardo, a psychologist, explains that it is due to the basic needs of people to be loved, respected, and recognized that they can choose or be induced to behave in a manner which will bring them affirmation [23]. There is abundant sociological evidence of conformity within groups [44]. Generally, individuals in groups conform to a greater degree if they have a strong desire to remain a member, if they are satisfied with the group, if the group is cohesive, and if they are not a natural leader in the group. Furthermore, the individuals are generally unaware that they have modified their judgments to be in agreement with the group.*

One mechanism for this unconscious modification of opinion is explained by the theory of cognitive dissonance. Cognitive dissonance occurs when individuals find a discrepancy between thoughts that they hold or between their beliefs and actions [45]. For example, if an individual holds an opinion that conflicts with that of the other group members and he or she has a high opinion of the other's intelligence, cognitive dissonance will result. Often the individuals' means of resolving the discrepancy is by unconsciously changing their judgment and/or the reporting of that judgment to be in agreement with that of the group [46]. For example, in American foreign policy presidential advisers often silently acquiesce rather than critically examine what they believe to be the group judgment [47]. This phenomena has also been called *group think bias*, the *follow-the-leader* or *bandwagon* effect.

Group think is only likely to be a concern in an interactive group situation. It is further likely to occur in situations where behavioral aggregation is used because this type of aggregation requires that pressures toward conformity be encouraged.

**Impression management** is another type of social pressure that can occur as the expert imagines the reactions of those not physically present. This effect can occur in any elicitation situation; however, it seems to be more noticeable when it is not covered by other effects, such as social pressure caused by the knowledge engineer. For this reason, its occurrence is most noted in mail surveys or Delphi situations. Individual may try to answer in such a way as to bring the most approbation, such as from the person who has written the questions. Then too, they may

---

* Note that we are making a distinction between unthinking conformity and experts changing their minds as a result of new information received. The former is group think; the later an illustration of knowledge in action.

try to respond in such a way as would be acceptable to their employers or to society in the abstract. For this reason, this type of social pressure has been termed impression management [48]. There is evidence of individuals giving the responses that they perceived to be the most socially acceptable rather than those which accurately portrayed their thoughts or actions [49]. For example, on surveys, people often claim that their educations, salaries, and job titles are 10% better than they are [43].

**Misinterpretation** is the altering of the expert's responses as a result of the methods of elicitation and documentation. While this effect is prevalent, it has not received much attention. Frequently misinterpretation occurs as a result of the response mode. If the experts cannot adequately translate their judgments into the response mode, misinterpretation will result. We have noticed that experts seem to have more difficulty with the following response modes: probability distributions, ranks, and percentiles.

Misinterpretation is also more likely with elicitation and documentation methods that are written from the knowledge engineer's, rather than the expert's, viewpoint. For example, we have all had the frustrating experience of trying to force fit our views into the limited response options of a mail survey.

**Wishful thinking** occurs when individuals' hopes influence their judgment [24]. What the subjects think should happen will influence what they think will happen. To illustrate, presidential election surveys show that people predict the winner to be the candidate for whom they expect to vote [50]. In this instance, the subjects gain very little personally from their answer. The wishful thinking effect is stronger where the subjects are personally involved or would gain from their answers. Thus, this bias is also called *conflict of interest*. People also exhibit wishful thinking about what they can accomplish in a given amount of time: they overestimate productivity [51].

Wishful thinking is not particular to any elicitation method. Instead it relates to the selection of experts and the assignment of them to specific questions or problems. If the experts have a special interest in the answer, wishful thinking is likely to occur whether the individual interview, interactive group or Delphi elicitation method is used or whether the communication is face-to-face, by telephone, or mail. For this reason, we suggest selecting those experts who have the least to gain from the response that they give. Often, however, those most qualified will also be those with the most at stake. For this reason, Boose and Shaw [52] advise obtaining data from experts holding divergent views. One approach to obtaining diverse expertise is to select experts that represent different organizations (e.g., government, academia, and industry), and the various theoretical stances on the subject.

In general, the effects of wishful thinking will be most pronounced when the experts do not have to explain their reasoning. The experts' highly optimistic responses are checked by having them disaggregate the problem and explain their problem solving. For example, Hayes-Roth [51] found that having people break down the tasks that they had earlier thought they could accomplish in a given time led to more realistic estimates.

**Cognitive Bias** occurs when data fails to follow mathematical and logical standards because of inconsistency, anchoring, availability, or underestimation of uncertainty.

**Inconsistency** is the inability to be consistent in one's solving of problems, especially through time. Of all of the biases mentioned here, this is the most common. Individuals often unintentionally change definitions, assumptions, or algorithms that they meant to hold constant throughout the problems. Inconsistency in individuals' judgments can stem from their remembering or forgetting information during the elicitation session. For example, individuals may remember some of the less spectacular pieces of information and consider these in making judgments later in the session, or they may forget that particular ratings were only to be given in extreme cases and begin to assign them more freely toward the end of the session.

As Dawes, Faust and Meehl [31] have noted, such factors as fatigue, recent experience, or seemingly minor changes in the ordering of the information or in the conceptualization of the "...task can produce random fluctuations in judgment. Random fluctuation decreases judgmental reliability and hence accuracy." These inconsistencies may result in answers that do not make logical or Bayesian sense. For instance, a series of answers proposing that factor A was more critical than B, B more than C, and C more than A would not make logical sense. Similarly, if an expert gave the same probability of A for two situations, one of which involved an influential factor C and one which did not, his or her answers would not be coherent from a Bayesian viewpoint.

The natural tendency toward inconsistency is acerbated by several conditions such as memory problems, fatigue, and confusion. During elicitation sessions of more than 30 minutes, people often forget the instructions, definitions, or assumptions that they were requested to follow. For example, the experts may forget that a rating of nine meant a near certainty and assign it more easily than the definition specified. Thus, unstructured elicitations, which do not have periodic reviews of the problem information, are more likely to have high inconsistency. This inconsistency can occur between experts' answers (e.g., the experts meant different things by the same numerical answer) or within an expert's answer (e.g., sometimes the expert gave a specific rating more easily than at other times). Also, situations where the expert's understanding through time cannot easily be monitored are more prone to inconsistency. These situations include the Delphi or mail survey.

Confusion can also lead to inconsistency. Thus, any of the more complicated response modes, such as probability distributions and percentiles, are more prone to this problem. Therefore, training the expert in the use of these modes (step 3) is recommended. In addition, if the experts must mentally juggle more than five to seven things, they are likely to become confused and inconsistent. It is for this reason that the Saaty paired-comparison mode [53] is used even though it is more time consuming than some of the other response modes.

**Anchoring** is the failure to adjust sufficiently from one's first impression in solving a problem. We would rate it next to inconsistency in terms of frequency of occurrence. Sometimes this tendency is explained as the failure to adjust a judgment in light of new information as much as it would be in terms of Bayes mathematical formula [43]. Spetzler and Stael von Holstein [54] and Armstrong [50] describe how people tend to anchor to their initial response, using it as the basis for later responses. Ascher [55] has found this problem to exist in forecasting where panel members tend to anchor to past or present trends in their projection of future trends. Ascher determined that one of the major sources of inaccuracy in forecasting future possibilities, such as markets for utilities, was the extrapolation from old patterns that no longer represented the emerging or future patterns. Another example of anchoring occurs when group members' last estimates are closer to their initial impressions than these would be had they fully taken earlier group discussions into account.

Anchoring is most prevalent in situations where the expert is not likely to experience the opposite bias of being influenced by the knowledge engineer or the group, such as in the Delphi method. In addition, those modes, such as mail or telephone communications, where the expert's thoughts cannot be easily monitored by having the expert think aloud, are prone to this bias. Also, we have noticed that experts are more likely to stick with their anchor if they have either described it orally or in writing and fear losing face for changing their mind.

**Availability** bias arises from the differing ease with which events can be retrieved from long-term memory. Data involving catastrophic, familiar, concrete, or recent events tend to be easier to recall. Contributing to this bias is our human tendency to think that "we know more than we do and also that what we don't know must be unimportant" [52]. Availability bias affects people's ability to accurately estimate frequencies and recall other aspects of the event. For example, the incidence of severe accidents in reactors tends to be overestimated in part because of their catastrophic and newsworthy nature.

Availability bias is more common when the expert does not receive any information from others and, thus, does not have a chance of triggering other, less accessible, memory associations. For this reason, the individual meeting is the most prone to availability bias, and the interactive group, the least. With individual meetings, a series of different scenarios is often used to help the experts enlarge the scope contributing to their final answers.

Availability bias is also more common with telephone and mail modes of communication because the expert is usually not given much background before being asked point blank for the answer. A structured hierarchical presentation of the information, such as from the general to the specific, can alleviate this weakness.

**Underestimation of uncertainty** occurs when people underestimate the amount of uncertainty in the answers that they give. For example, when people are asked to put a range around an answer such that they are 90% sure that the range encompasses the correct answer, their ranges only cover about 58% of the probable range of values [56]. Even when they are given quizzes and feedback on their performance, they cannot break the barrier of covering only 70% [56]. A popular explanation for this effect is that we are uncomfortable with the amount of uncertainty in our lives, and thus, minimize it. In particular, we may avoid confronting the large uncertainties in our judgments. Also, in planning activities, we may underestimate uncertainty because of the chain-like nature of planning. We may ignore factors which could cause a delay because each factor seems unlikely by itself [52].

Although underestimation of uncertainty is very widespread, it is more pronounced with probability and chance estimates than with some of the other response modes [57] . Chance estimates, also called *odds*, are given as 1 chance in a total, such as 1 in 1000.

### *2.2.2. Step 2--Redesign the planned elicitation to make it less prone to the anticipated biases*

If the knowledge engineers anticipate bias, the definition section mentioned above can be used to redesign the elicitation. For instance, suppose that the knowledge engineers became concerned with the potential for wishful thinking bias as a result of reading the *Index* and *Definitions* in step 1. They could turn to the definition of wishful thinking and replan their strategy for sampling the experts. In the *Definition* section, the knowledge engineers would receive the suggestion to select those experts who have the least to gain from their responses, or, alternatively, to sample experts from different organizations and schools of thought to obtain diverse views on the subject.

In addition, some of the suggestions given in step 5 for countering selected biases (section 2.2.5.1.) could be integrated in the designing of the elicitation. For example, one of the ways for countering wishful thinking is to ask the experts to explain their thinking in detail. The knowledge engineers could incorporate this requirement into their elicitation design (e.g., plan for problem disaggregation with the experts explaining their reasoning on each).

### *2.2.3 Step 3--Make the experts aware of the potential intrusion of particular biases and familiarize them with the elicitation procedures*

The experts need to be (1) informed about the biases, and (2) acquainted with the elicitation procedures. They need to know bias definitions and reasons for occurrence. Without this information, the experts will not be able to combat their own tendencies toward bias. The

knowledge engineer can use the *Index* and *Definitions* provided for step 1 as a base for informing the experts about bias.

It should be noted that making the experts aware of the biases helps but does not completely alleviate the problem. For example, if experts are aware of how their responses may be misinterpreted or influenced by the knowledge engineer, they can help monitor for these biases. However, in some cases the cause of the bias, such as the experts' underestimation of uncertainty, is too ingrained to be completely overcome. In other cases, the experts will not make the needed effort to counter the natural tendency toward bias. People typically believe that others, not themselves, will suffer from the biases described. With some biases, such as anchoring and underestimation of uncertainty, the experts can participate in tests designed to evoke the bias. Test questions with verifiable answers are used, such as: *How much rain fell in St. Paul in 1987?* The knowledge engineer can read the answers and allow the experts to score themselves. Such a demonstration is often necessary to convince the experts that each of them are prone to the bias.

The experts also need to be made aware of the elicitation procedures. If they are confused about how and when they are to respond, the data gathered as well as the expert's cooperation may be negatively affected. The use of unfamiliar response modes should be rehearsed by the experts during the training session. [If the response mode is probability distributions, Hogarth [24] provides eight keys to its use.]

*2.2.3.1. Familiarize the experts with the elicitation procedures.* Information on how to familiarize the expert with the elicitation procedures is given below by elicitation situation.

For an individual meeting

1. Give the experts some sample questions to familiarize them with the use of the response mode, if that mode is likely to be a difficult one for them (i.e., if they do not use it as a part of their normal problem solving).
2. Brief the experts on any biases that were identified as being likely to occur (step 1). Give the experts ideas on how they can strive to counter the tendency towards these biases. (The section Definitions of Selected Biases in section 2.2.1.3. provides examples of this type of information.)
3. Give the experts the set of questions and verbally go over any instructions. Ask if they have any questions.

For an interactive group meeting

1. Review the purpose of the project, its schedule, and, in general, the elicitation procedures for the benefit of the experts. Some descriptions of the elicitation procedures are as follows: *You will meet together for this week to develop detailed statements or representations of the problems. On the last day, Friday, you will vote on what you think the answers should be.* A more detailed overview of elicitation procedures is as follows: *You will meet here three times. First, to become familiar with the project and the elicitation procedures; second, to present up-to-date technical information and refine the rough-draft questions; and third, to give your expert judgment in private meetings with a knowledge engineer.*
2. Give the experts sample questions to work so that they can practice using the response mode. If there are any techniques to properly using the response mode, they can be introduced and practiced here.
3. Brief the experts on those biases that were identified in step 1 as being likely to occur. This briefing should include an explanation of why the selected biases occur and of how the experts can reduce their tendencies to introduce them. (See the Definitions section.) In addition, the briefing should include exercises that are designed to evoke the selected biases. After the experts have completed the exercises, the moderator/knowledge engineer can read the correct answers and allow the experts to correct their own. These exercises can convince the experts that they, like everyone else, are prone to these biases.
4. Ask if there are any questions. Afterwards, state that the introduction is concluded and the elicitation sessions will now begin.

For a Delphi situation

- If the entire Delphi will be conducted by mail, the expert will not be introduced to the elicitation through the mail by receiving a hard copy of the set of questions and a cover letter.
- If part of the Delphi will be conducted by telephone, call the experts to assist them in understanding the set of questions or just to obtain their answers. Use the items listed for individual meetings above as a basis for introducing the experts to the elicitation process.

### *2.2.4. Step 4--Monitor the elicitation for the occurrence of bias*

Prior to the elicitation sessions, the knowledge engineer reads about the signs that signal the occurrence of particular biases (See *Signs of Selected Biases* below.) For instance, if group think bias was anticipated, the knowledge engineer would look up this bias in the *Signs* section and read about indications of its presence. One sign of group think is that the experts appear to defer to another member of the group or to each other. The knowledge engineer or a trained observer then watches for this sign of group think during the elicitation. In general, monitoring biases, as

described here, requires that the experts verbalize their thoughts and answers. Without this feedback, we have found the monitoring to be much more difficult.

*2.2.4.1. Signs of selected biases*

**Group think**, as it develops, may be indicated by several signs. Generally, no difference of opinion is voiced, and the experts appear to defer, verbally or behaviorally, to another member of the group or to each other [43].

**Wishful thinking** may be difficult to detect because its signs are subtle. Watch for experts responding before they have had time to think through the question or for experts unduly interested in their managements' access to their responses.

**Inconsistency** is indicated by a number of signs. The knowledge engineer can hear many of these, if the experts are verbalizing their thoughts and answers. In particular, the knowledge engineer can detect when a rating is being applied more easily through time [43] because experts tend to apply the extremes of a scale as they become fatigued. Knowledge engineers can also hear when the experts are contradicting an assumption that they made earlier.

Inconsistency can also be monitored by the use of Bayesian-based scoring and ranking techniques.such as Saaty's Analytical Hierarchical Process [53], to obtain an index of their consistency. Then, if this number is too high, indicating significant inconsistency, the knowledge engineer can ask the experts to redo their judgments.

**Availability** may be indicated if the experts do not mention more than one or two considerations in giving their answers. If the experts only considers a few things, these were probably the most easily remembered and the answer is likely to be skewed to reflect these few.

**Anchoring** can be suspected if the experts receive additional information from experts or other sources during the elicitation but never waiver from their first impression. For example, reactor-code modeling experts were asked to compare the performance of their computer models to plots of experimentally generated data. Often they commented on their first impression. When they examined the plots more closely, they typically found places where the computer model did not capture the experimental phenomena. However, the experts usually simply adjusted upward or downward of their initial assessment rather than revising it completely [40].

*2.2.5. Step 5--Adjust, in real time, to counter the occurrence of bias*

In this step, the knowledge engineer looks up ways for preventing a particular bias in *Suggestions for Countering Selected Biases* below. These suggestions vary because we have used two approaches: (1) controlling those factors that contribute to a particular bias, or (2) employing the opposite bias. For instance, using the first approach, the knowledge engineer would consider that fatigue is a factor leading to increased inconsistencies in expert judgment. As a control, the knowledge engineer could stop the elicitation sessions or schedule breaks before the experts become fatigued. The basis of the second approach, fighting bias with bias, comes from Payne [49], the grandfather of survey design. Payne believed that all interviewing was biased and that one should therefore aim for equal but opposite biases. An example of this technique would be to encourage the experts towrite down their individual judgments prior to discussion in attempts to counter a group-think situation.

This step is perhaps the most delicate one because if done carelessly it could affect the analysis and confuse the results. The knowledge engineer needs to decide, in advance, on the timing of the adjustment because these adjustments change the conditions under which the data is gathered. A condition leading to bias needs correcting before the analyzable data has been gathered. If the analyzable data has been gathered under two conditions, when bias was occurring and then when it was corrected, the data will be mixed. Unless the two situations can be clearly separated, the knowledge engineer will not be able to later test for the presence of this bias.

*2.2.5.1. Suggestions for countering selected biases*

**Group think**, as a type of social pressure, can be countered using techniques from both the approaches (1) and (2) above [43]. Using the first approach, the knowledge engineer can stop the elicitation and warn the group members about group think. If there are official or even natural ex officio leaders in the group, they can be asked to give their responses last, or privately, so as not to influence the other group members. In addition, if persons other than the knowledge engineers have been leading the group meeting, they can be encouraged to be nondirective during the meetings. An explanation of the group-think phenomena usually convinces them that better discussions and data will result from their avoiding *leading*.

The other approach is to try to counter the effects of group think with anchoring. One technique for fostering anchoring is to require the group members to write down their judgments and reasoning. In this way, they are more likely to anchor to their own judgments rather than silently acquiesce to someone else. If the experts are to discuss their judgments, each of them can record and report their positions before the floor is opened to discussion. Once individuals have publicly announced their view, they are unlikely to spontaneously modify it.

**Wishful thinking** can be countered by making it more difficult for the expert to indulge in it. If the experts must explain their answers in detail, it will become apparent whether there was any basis beyond the experts' motivations

for their responses. If the experts are providing estimate of their own performance, Boose and Shaw [52] recommend focusing the discussion on contingencies that are outside the experts' control but may affect their achievements.

**Inconsistency** can be reduced by using several techniques. The first is to address the aspects of the elicitation that are contributing to the inconsistency. As mentioned earlier, fatigue is a contributor to inconsistency. If the knowledge engineer has noted that the experts are becoming more inconsistent with time, she or he can quickly end the meeting or schedule a break. In general, two hours is the maximum amount of time that experts participate in discussion or problem solving before becoming tired.

Another contributor to inconsistency is faulty memory. If at the beginning of every session the statement of the problem, definitions, assumptions, and response mode are reviewed, the experts will be more consistent in their judgments [43]. They will be more consistent between and within themselves. In addition, if there is much time between this first review and when the experts' judgments are requested, the question can be worded to include some of the above information. For example, "What rating would you give to the importance of element *x* over *y* to the reaching of goal *z*?" If they are using a response mode, in this case a Saaty paired comparison, they will need to have the definitions of the comparison scale available in front of them.

Another technique for reducing inconsistency is to have the group members monitor their own consistency [43]. This technique was successfully used in a simple interactive group elicitation where the experts were able to watch the knowledge engineer's monitoring of inconsistency and then mimic it [58]. The experts were given copies of a matrix of the elements being judged, the criteria on which they were being judged, and their past judgments. When experts monitor their own consistency, they may wish to change an earlier judgment to be in line with their current thinking. If their reasoning does not violate the logic of the model or the definitions, they can be allowed to make the change. Often in this process, the experts may discover that they had forgotten to include some pertinent information. After this addition, some of the judgments may need to be redone.

**Availability** bias can be countered by stimulating the expert's memory associations. In general, group discussion will cause the expert to think of more than just the first readily accessible information. In addition, free association can be introduced to single experts or those in groups. Free association is having the expert or experts quickly generate any and all elements that might have bearing on the question [43]. Free association is similar to brainstorming or the Crawford Slip method [33]. The experts are asked to refrain from being critical in order to generate the widest possible pool of ideas. (This list is later narrowed to those judged to be most pertinent to the problem.) A related technique is to hierarchically structure the presentation of problem information so that it flows from the general to the specific. In this way, the expert is able to consider the pertinent information before having to reach a solution. Again, this strategy is to fire as many memory associations as possible so that the maximum number of relevant ones will enter into the expert's final judgment.

Three additional ideas for countering availability bias from Boose and Shaw [52] are: (1) to ask the experts to describe how other experts might disagree with their responses, (2) to ask the experts to temporarily forget recent events, and (3) to aggregate outcomes with small probabilities into a single larger class to reduce the perceived joint impact if probability estimates are being elicited.

**Anchoring** can be countered by some of the techniques given for availability bias. In particular, giving the experts input from others as in a Delphi situation or an interactive group makes it more difficult for them to anchor to first impressions. Another technique is to ask the experts for extreme judgments before getting their likely ones [28].

**Underestimation of uncertainty** may be reduced by asking the expert to further disaggregate the parts of the question and give estimates of the quantities of interest for each part. In this way, the experts are less likely to overlook something; and the knowledge engineers can check whether the details of each of the experts' thinking correctly add up to their answers. If the underestimation is occurring in a planning problem, a comprehensive list of events that could upset the plans can be elicited from the expert [52]. Creating this list may make the experts realize that they should take some of these possibilities into account in their problem solving.

### *2.2.6. Step 6--Analyze the data for the occurrence of bias*

This step can be conducted at two different levels: (1) simple analyses and (2) complex analyses.

*2.2.6.1. Simple analyses.* The first level is often used if step 6 is the only one that the knowledge engineer will be doing. In addition, these simpler tests are frequently used on cognitive biases because they are more amenable to straightforward analyses.

However, even doing this simpler level of analysis, presupposes that sufficient data was gathered. One of the major problems in testing for bias is having a sufficient sample size. The sample can be composed of multiple experts' data on the same problem or a single expert's data on many problems. To ensure that there is enough analyzable data, it is helpful to consult with a statistician before beginning the elicitation. In addition, we would caution the persons performing the analysis to beware of misrepresenting the data in analyzing it. The use of many statistical techniques require that the analyst make particular assumptions about the data. If these assumptions are not warranted, the data will be misrepresented, biased according to the

motivational definition. For example, many statistical techniques require that the analyst assume that the data is normally distributed. Our experience has been that the data is multimodal and not normally distributed. Nonparametric statistical procedures and data-based simulation techniques are recommended wherever possible because they do not require distribution assumptions.

Two examples of testing at a simple level are given below.

**Group think** bias was tested for in a project whose goal was to elicit estimates on the relevance of selected weapon-related components to particular military needs [58]. The experts had assigned weights to describe how a weapon would meet a defense need. The weights ranged from 0, meaning not at all related, to 3, meaning 80% to completely related. Group think had earlier been judged a possibility because the experts were from the same work group, had worked together before, and had been selected for this project by their supervisor, who was present to lead some of the sessions. In addition, behavioral aggregation was being used, which further predisposed the elicitation toward group think. Thus, it was possible that the leader's views were influencing the discussions and the assignment of weights. To test for this possibility, the estimates were grouped according to who had led the session and analyzed for correlations to the weights. There was no evidence of this effect, but the conclusion was not absolute because some of the factors overlapped making definitive testing of their effects impossible.

**Underestimation of uncertainty** was the focus of analysis in another project. The experts had estimated the likelihood of achieving national magnetic fusion milestones within particular time periods. This bias was considered likely because people typically underestimate uncertainty; and therefore, the time and money needed to complete various tasks. In addition, people seem to be more prone to this underestimation when they are estimating on uncertain situations, such as fusion technology. Research by Rand Corporation has shown that cost estimates for a first-of-a-kind project display greater cost errors than for other kinds of projects [59]. For these reasons, the expert's ranges on a question were analyzed. They were found to be within one standard deviation of the set of answers. This indicates that the experts thought that they were adequately accounting for uncertainty when they were accounting for less than 60% [22].

*2.2.6.2. Complex analyses.* The second level of analysis involves using the data gathered from monitoring and adjusting for bias (steps 4 and 5) to do a more complete analysis. Information gathered during these steps, such as who appeared to be exhibiting a particular bias or differed in their problem-solving processes, can offer insights into what biases to test for. In addition, if the later analysis yields confusing results, this data can aid in their interpretation.

For conducting the more complex analyses, we recommend the use of multivariate and simulation techniques. Multivariate techniques allow the simultaneous consideration of two or more variables of interest. This capability is important because expert data is frequently conditioned on other data. An expert's data can be conditioned on such factors as aspects of the elicitation situation, the expert's interpretation of the question, and method of solving the problem. Because of the small sample sizes usually associated with knowledge acquisition, these effects can be indistiguishable in the analysis. For example, in the early project mentioned above, we were not able to definitively test for group think because we could not separate this variable from others.

We have used several multivariate techniques, such as cluster analysis, discriminant analysis, and general linear models in trying to find the one most applicable to the data in question. If these multivariate techniques require assumptions that are not applicable to the data, simulation techniques can be used. Simulation techniques can also be used to discover relationships between the variables without requiring unreasonalble assumptions. A statistician can provide guidance in selecting the specific analytic techniques.

In two studies, tests were performed at this more complex level. In these studies, we searched for sources of correlation, or dependence, between expert's solutions [40]. The experts solved problems in individual meetings. Data was gathered on aspects of the expert's professional backgrounds, the problems, and the expert's means of solving the problem. This data was combined in a large data base, and its multivariate relationships were analyzed using multivariate and simulation techniques.

**Anchoring bias,** in another project, was examined using an experimental design. The experts in this project received limited information from individuals threatening violence. They had to decide if the communication was a ploy or if it should be taken seriously. During the monitoring of the first set of elicitations (step 4), the knowledge engineer discovered that the experts were making implicit assumptions. Follow-up sessions indicated that about half of the experts assumed that the communications were ploys until proven otherwise; the other half assumed that they were credible until proven otherwise. For the problem-solving sessions, experts were assigned to questions so a pair of each assumption type would be represented by a problem. With such a design and the earlier mentioned techniques, it was possible to test for the effect of this assumption on the expert's final conclusion.

## 3. Summary

An approach for treating bias has been proposed here. The approach includes guidance on when to use particular combinations of steps depending on the knowledge engineer's definition of bias, reasons for focusing on bias, and interest in particular types of bias. The steps are (1) anticipate which biases are likely to occur in the planned elicitation; (2) redesign the elicitation to make it less prone to the anticipated biases; (3) make the experts aware of the potential intrusion of these biases and familiarize them with the elicitation procedures; (4) monitor the elicitation for the occurrence of these biases; (5) adjust, in real time, to counter their occurrence; and (6) analyze the data to determine more definitively if they occurred. The steps can be followed in sequence or singly, depending on the needs of the knowledge acquisition task. The approach describes in detail how to perform each step using selected biases as examples.

While this program was developed for manual knowledge acquisition, it could be applied in part to automated, or interactive, acquisition. In automated knowledge acquisition, there will still be a need for the knowledge engineer, according to Boose and Gaines [33]. Thus, the knowledge engineer could, in between assisting experts with the automated tools, follow the steps for countering bias. Furthermore, the automated tools could assist the knowledge engineer in detecting and countering bias, especially those covered by the cognitive definition.

In this paper, we hope to make knowledge acquisition practitioners more aware of bias, their options in defining it, and their potential contribution to the problem. We aim to provide a more explicit, disciplined approach to countering bias in elicitation; such an approach benefits not only manual knowledge acquisition but creates a foundation for automating the countering of bias. Through this paper, we hope to stimulate other work on handling bias in knowledge acquisition and to obtain feedback for developing our program.

We are indebted to the participants of the 4th Knowledge Acquisition Workshop, Banff, for comments on the first draft of this paper. In particular, Hans Acckermans and Stephen Regoczei generously contributed their time and ideas for developing this paper. The financial support of the Nuclear Regulatory Commission (FIN #A7225) for the writing of the report (NUREG/CR-54-24) from which this paper was developed is gratefully acknowledged.

## References

[1] ORTIZ, N. R., WHEELER, T. A., MEYER, M. A., AND KEENEY, R. L. (1988). The use of expert judgment in NUREG-1150. *Proceedings of the 16th Water Reactor Safety Information Meeting*. Oct., Gaithersburg, MD. In Nuclear Regulatory Commission Report NUREG/CP-0097, Vol. 5, pp. 1-25.

[2] SHAFER, G., & TVERSKY, A. (1985). Languages and designs for probability judgment. *Cognitive Science*, **9**, 309-339.

[3] GAINES, B. R. (1989). Social and cognitive processes in knowledge acquisition. *Knowledge Acquisition*, **1**, 39-58.

[4] BENBASAT, I. & DHALIWAL, J. S. (1989). A framework for the validation of knowledge acquisition, *Knowledge Acquisition* , **1**, 215-233 .

[5] KELLY, G.A. (1955).. *The Psychology of Personal Constructs.* 2 Volumes. New York, NY: Norton.

[6] ADAMS-WEBBER, J.R. (1989). Kelly's pragmatic constructivism. *Canadial Psychology*, **30**, 190-193.

[7] AGNEW, N.M., & BROWN, J.L. (1989). Foundations for a theory of knowing: I. Construing reality. *Canadian Psychology*, **30**, 152-167.

[8] FORD, K.M. (1989). A constructivist view of the frame problem in artificial intelligence. *Canadian Psychology*, **30**, 188-190.

[9] BATESON, G. (1979). *Mind and Nature: A Necessary Unity*, New York, New York, NY: Bantam Books.

[10] BRADSHAW, J. M. & BOOSE, J. H. (1990). Decision analysis techniques for knowledge acquisition: Combining information and preferences in Aquinas and Axotl. *International Journal of Man-Machine Studies*, **32**, 121-186.

[11] SHAW, R. & MCINTYRE, M. (1974). Algoristic foundations to cognitive psychology. In W. B. WEIMER & D. S. PALERMO, Eds., *Cognition and the Symbolic Processes*. New York, NY: Erlbaum.

[12] LANGER, E. J. (1989). *Mindfullness*, Reading, MA: Addison-Wesley.

[13] DOUGLAS, M. (1970). *Purity and Danger: An Analysis of Concepts of Pollution and Taboo*, London, England: Routledge and Kegan Paul.

[14] WEIMER, W.B. (1979). *Notes on the Methodology of Scientific Research.* Hillsdale, NJ: Erlbaum.

[15] FELDMAN, B. Z., COMPTON, P. AND SMYTHE, G. A. (1989). Hypothesis testing: An appropriate task for knowledge-based systems, *Proceedings of the 4th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, October, pp. 10-1 to 10-20.

[16] KAPLAN, A. (1963). *The Conduct of Inquiry.* New York, NY: Harper and Row.

[17] ADAMS-WEBER, J.R. (1981). George A. Kelly--An appreciation, in M. L. G. Shaw, Ed., *Recent Advances in Personal Construct Technology*, pp. 1-7. London, England: Academic Press.

[18] HENRION, M. & COOLEY, D. R. (1987). An experimental comparison of knowledge engineering for expert systems and for decision analysis, *Proceedings of the American Association for Artificial Intelligence*, Los Altos, CA: Morgan Kaufman Publishers, pp. 471-476,

[19] MUMPOWER, J. L., PHILLIPS, L. D., RENN, O. & UPPULURI, V. R. R. (1987). *Expert Judgment and Expert Systems*, NATO ASI Series, New York, NY: Springer-Verlag.

[20] BEYTH-MAROM, R. & DEKEL, S. (1985). *An Elementary Approach to Thinking Under Uncertainty*. Translated and adapted from Hebrew by S. Lichtenstein, B. Marom, & R. Beyth-Marom. Hillsdale, NJ: Lawrence Erlbaum.

[21] KAHNEMAN, D., SLOVIC, P. & TVERSKY, A. (1982). *Judgment Under Uncertainty: Heuristics and Biases*. Cambridge, MA: Cambridge University Press.

[22] MEYER, M. A., BOOKER, J. M., CULLINGFORD, H. S. & PEASLEE, A. T., JR. (1982). A Data-Gathering Method for Use in Modeling Energy Research, Development and Demonstration Programs, *Energy Programs, Policy, and Economics: Alternative Energy Sources*, pp. 421-430. Ann Arbor, MI: Ann Arbor Science.

[23] ZIMBARDO, P. G. (1983). To Control a Mind, *Standford Magazine*, Winter, 59-64.

[24] HOGARTH, R. M. (1980). *Judgement and Choice*, New York, New York: Wiley and Sons.

[25] DENNING, P. J. (1986). The science of computing: will machines ever think? *American Scientist*, **74**, 344-346.

[26] BREUKER, J. & WIELINGA, B. (1988). Models of expertise in knowledge acquisition. To appear in G. GUIDA AND C. TASSO, Eds., *Topics in Expert System Design: Methodologies and Tools*. Amsterdam, The Netherlands: North Holland Publishing Co.

[27] MILLER, G. A. (1956). The magical number seven, plus or minus two: some limits on our capacity for processing information, *Psychological Review*, **63**, 81-97.

[28] CLEAVES, D. A. (1986). Cognitive bias and corrective techniques: proposals for improving elicitation procedures for knowledge-based systems, in *Proceedings from the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, pp. 9-0 to 9-11.

[29] GOLD, P. E. (1987).Sweet memories, *American Scientist*, **75**, 151-155.

[30] FAUST, D. & ZISKIN, J. (1988). The expert witness in psychology and psychiatry, *Science*, **241**, 31-35.

[31] DAWES, R. M., FAUST, D. & MEEHL, P. E. (1989). Clinical versus acturial judgment, *Science*, **243**, 1668-1673.

[32] ROSENFIELD, I. (1988). *The Invention of Memory: A New View of the Brain*, New York, NY: Basic Books.

[33] BOOSE, J. H. & GAINES, B. R. (1988).Knowledge acquisition for knowledge-based systems, *Tutorial MP4 of the Seventh National Conference on Artificial Intelligence of the American Association for Artificial Intelligence*, St. Paul, MN, August .

[34] MCGRAW, K. L. & HARBISON-BRIGGS, K. (1989) *Knowledge Acquisition: Principles and Guidelines*, Englewood Cliffs, NJ: Prentice-Hall.

[35] MEYER, M. A. & JOHNSON, E. R. (1985). Proceedings from the Technology Control Panel workshop, Los Alamos National Laboratory Technical Report LA-UR-85-4078, Los Alamos, NM.

[36] MEYER, M. A., MNISZEWSKI, S. M. & PEASLEE, A. T. JR. (1989). Using three minimally biasing elicitation techniques for knowledge acquisition, *Knowledge Acquisition*, **1**, 59-71.

[37] WIELINGA, B., AKKERMANS, H., SCHRIEBER, G. & BALDER, J. (1989). A knowledge acquisition perspective on knowledge-level models, *Proceedings of the 4th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, October, pp. 36-1 to 36-22.

[38] SHAW, M. L. G. & WOODWARD, J. B. (1989). Mental models in the knowledge acquisition process, *Proceedings of the 4th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, October, pp. 29-1 to 29-24.

[39] DHALIWAL, J. S. & BENBASET, I. (1989). A framework for the comparative evaluation of knowledge acquisition tools and techniques, *Proceedings of the 4th Annual Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, October, pp. 9-1 to 9-24.

[40] MEYER, M. A. & BOOKER, J.M. (1989). Eliciting and analyzing expert judgment: A practical guide, Nuclear Regulatory Commission Report NUREG/CR-5424, Washington, D.C.

[41] SEAVER, D. A. (1976). Assessments of group preferences and group uncertainty for decision making, Social Science Research Institute, University of Southern California, Los Angeles, CA.

[42] REGOCZEI, S. & HIRST, G. (1989). Sortal analysis with SORTAL: A Software Assistant for Knowledge Acquisition, *Proceedings of the 4th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, October,.pp. 25-1 to 25-18.

[43] MEYER, M. A. (1986). Human ractors affecting subjective judgments, *Proceedings of the Thirty-First Conference on the Design of Experiments in Army Research Development and Testing*, Madison, WI, pp. 85-98.

[44] WEISSENBERG, P. (1971). *Introduction to Organizational Behavior: A Behavioral Science Approach to Understanding Organizations*, Scranton, OH: Intext Educational.

[45] FESTINGER, L. (1957). *A Theory of Cognitive Dissonance*, Palo Alto, CA: Standford University Press.

[46] BARON, R. A. AND BRYNE, D. (1981). *Social Psychology: Understanding Human Interaction*, Boston, MA: Allyn and Bacon.

[47] JANIS, I. C. (1972). *Victims of Group Think: A Psychological Study of Foreign Policy Decisions and Fiascos*, Boston, MA: Houghton Mifflin.

[48] GOFFMAN, E. (1959). *The Presentation of Self in Everyday Life*, Garden City, NY: Doubleday Anchor Books.

[49] PAYNE, S. (1951). *The Art of Asking Questions*, Princeton, NJ: Princeton University Press.

[50] ARMSTRONG, J. S. (1981). *Long-Range Forecasting: From Crystal Ball to Computer*, New York, NY: Wiley-Interscience.

[51] HAYES-ROTH, B. (1980). Estimation of time requirements during planning: interactions between motivation and Cognition, Rand Corporation ReportN-1581-ONR, Santa Monica, CA.

[52] BOOSE, J. H. & SHAW, M. L. G. (1989). Knowledge acquistion for knowledge-based systems, *AAAI-89 Tutorial Notes of the Eleventh International Conference on Artificial Intelligence*, Detroit, Minnesota, August.

[53] SAATY, T. L. (1980). *The Analytic Hierarchy Process: Planning, Priority Setting, and Resource Allocation*, New York, NY: McGraw-Hill.

[54] SPETZLER, C. S. & STAEL VON HOLSTEIN, C-A. S. (1975). Probability encoding in decision analysis, *Journal of the Institute of Management Science*, 22, 340-358.

[55] ASCHER, W. (1978). *Forecasting: An Appraisal for Policymakers and Planners*, Baltimore, MD: John Hopkins University Press.

[56] CAPEN, E. C. (1975). The difficulty of assessing uncertainty, presented at the Society of Petroleum Engineers AIME 50th Annual Fall Technical Conference, Dallas, TX, September.

[57] MARTZ, H. F., BRYSON, M. C. & WALLER, R. A. (1985). Eliciting and aggregating subjective judgments--Some experimental results, Nuclear Commission report, NUREG/CP-0063, Washington, D.C.

[58] MEYER, M. A., PEASLEE, A. T., JR. & BOOKER, J. M. (1982). Group consensus methods and results, Energy Systems Group, Los Alamos National Laboratory Technical Report LA-9584-MS, Los Alamos, NM.

[59] MERROW, E. W., CHAPEL, S. W. & WORTHING. C. (1979). A review of cost estimation in new technologies: Implications for energy process plants, Rand Report R-2481-DOE, Santa Monica, CA.

# Methodological Foundations of Keats, The Knowledge Engineer's Assistant[1]

Enrico MOTTA, Tim RAJAN[2],
John DOMINGUE, and Marc EISENSTADT
Human Cognition Research Laboratory
The Open University
Milton Keynes, MK7 6AA, UK

**Abstract:** The numerous tasks required by the knowledge engineering process and their inherent complexity combine to make building knowledge-based systems both a time consuming and arduous activity. The key to reducing the complexity of the problem is to provide a methodological framework which can clarify the nature of the intermediate steps required to effectively encode knowledge into a performance system. Such a framework can then be used to drive the design of a comprehensive knowledge engineering toolkit. This is the approach we adopted in the Keats project. In this paper, we provide an overview of the Keats knowledge engineering methodology, which is based on a view of knowledge engineering as iterative refinement of qualitatively and teleologically different models, and we show how these ideas have driven the design of the Keats toolkit.

## 1. Introduction

The construction of large, knowledge-based applications is a complex task that comprises a number of activities and involves various participants. Knowledge engineers need to acquire the relevant knowledge from the relevant community [1], understand the domain in hand, design, implement, and debug the end system. Project managers are concerned with specifying and monitoring the project's activities, while end-users and domain experts are normally involved in knowledge acquisition sessions, and may be required to validate the content and the usability of the final system. It is therefore important that comprehensive knowledge engineering methodologies should be developed, in order to clarify the nature and the interrelations between these various activities, and provide guidelines for the various players involved in a knowledge-based project.

The efficacy of a knowledge engineering methodology is diminished if it is not supported by software tools. An ideal knowledge engineering toolkit should be able to give integrated software support to the various participants involved in knowledge-based projects, and across the whole range of tasks specified by a methodology. Unfortunately, this is not the case with the toolkits and shells that are currently available. Most of them assist the knowledge engineer only during the implementation and debugging stages of the knowledge engineering process. Even when a more comprehensive set of tools is provided [1], they do not afford overall life-cycle support.

Summarizing, our ideal toolkit should provide *principled*, *comprehensive*, and *integrated* assistance. A principled toolkit implements a systematic approach to knowledge engineering by instantiating a methodology for developing knowledge-based systems. Comprehensive assistance

[1] The authors gratefully acknowledge the support of British Telecommunications, plc. Discussions with Willem Jonker, Jyrki Kontio, Philippe Laublet, and Nigel Shadbolt helped us to improve the definition and the expression of our ideas.

[2] Tim Rajan's present address is: Symbolics Ltd, St John's Court, Easton Street, High Wycombe, HP11 1JX, UK.

implies that the whole range of activities defined by the methodology is supported. Integration requires that if tool A produces an output M, which is used by tool B to generate N, then it should be possible not only to navigate from M to N and vice versa, but also to enforce a consistency constraint between M and N, when required. In this case any change to M should be reflected in N and vice versa.

In the course of the Keats (The Knowledge Engineer's Assistant) project [2, 3, 4, 5, 6, 7, 8], we have tried to develop a toolkit that could fullfill this goal of providing principled, comprehensive, and integrated support for knowledge engineering. In this paper we give an overview of the methodological framework underlying the Keats architecture and we show how the various tools support the stages of the Keats methodology. The implementation described in this paper runs on the Symbolics 36xx family of workstations under Genera 7.2 and 7.4, and on compatible hardware such as MacIvory. The tools are now being ported to run on a wide range of Unix/Common Lisp/X-Windows platforms.

## 2. A model of the knowledge engineering process

### *2.1. Knowledge engineering methodologies and life-cycle models*

Methodologies for building knowledge-based systems typically attempt to provide life-cycle models of software development [9], and therefore focus on the managerial, technological and environmental aspects related to knowledge-based projects. This approach is similar to that adopted in software engineering [10], where prescriptive frameworks for managing software design and maintenance are outlined. The main advantage of this paradigm is that it provides models that can be effectively used to conduct and monitor knowledge-based projects, thus addressing the problem faced by project managers. Unfortunately, traditional life-cycle models give little assistance on how to tackle typical knowledge engineering problems such as choosing knowledge elicitation techniques, structuring the acquired data, mapping problem-solving strategies to tasks, choosing how best to represent a domain, designing the architecture of the system, and keeping track of the interrelationships among the outputs generated by these various activities.

The Keats project is an attempt to provide methodological and software support to help knowledge engineers tackle these problems. We view knowledge engineering as a process of information transformation, in which knowledge is acquired and ultimately transformed to a formal representation, resident inside a machine. We call this process of transformation *model refinement*, as each of the knowledge engineering stages produces a qualitatively different model of some aspects of the chosen *problem area*. For instance, models can be characterized at the *linguistic*, *conceptual*, *design*, or *implementation* level. Moreover, these models are teleologically different as, during the knowledge engineering process, they may be used for a number of purposes such as understanding the domain, documenting it, or effectively solving the task in hand. For the sake of terminology, we say that Keats provides a *knowledge engineering methodology*, aiming to describe the nature of the knowledge-transformation activities which the knowledge engineer has to perform when building knowledge-based systems. Therefore, the Keats methodology and toolkit primarily address the needs of knowledge engineers, rather than those of project managers, domain experts, or end-users. The methodology is also, to a large extent, independent of particular knowledge acquisition or knowledge representation techniques. In the rest of this paper we shall use the term *life-cycle models* to refer to approaches that, in contrast to Keats, are centred around models of management of knowledge-based projects.

### *2.2. A coarse-grained view of knowledge engineering*

At a coarse grained level, our analysis is centred around the three general classes of activities that knowledge engineers perform when building expert systems: *problem conceptualization*,

*knowledge encoding* and *debugging*. These are shown in figure 1, with ovals indicating processes and rectangles indicating data, models, or representations which are output from or input to one or more processes. As figure 1 shows, debugging is carried out both at the conceptual and at the system level, and is orthogonal to the other knowledge engineering tasks.

Problem conceptualization produces a *conceptual model* of the problem area. A problem area is characterized by a task, a domain's phenomenology, and the expert's problem solving behaviour. For example, if the problem in hand is one of diagnosing electronic components, the task is fault diagnosis, the domain's phenomenology concerns the structural, functional, and behavioural properties of electronic devices, and the problem solving behaviour is expressed by the heuristics and strategies that experts employ to perform diagnosis.

During knowledge encoding the architecture of the emerging knowledge-based system is designed and implemented. Debugging includes the validation, verification, and refinement of the models produced at the conceptual and at the knowledge encoding stage. As the issues, the methodologies, and the problems associated with maintaining a knowledge base are closely related to those involved in debugging, these two aspects of the knowledge engineering model will be discussed together.

A conceptual model is an abstract, machine-independent representation of the problem area. Typical representations used for conceptual modelling include taxonomic hierarchies, relational networks, tables, and flow diagrams. In the Keats methodology conceptual models play a role which is different from that played by the knowledge base. The former constitutes a conceptually adequate[1] representation of the problem and the domain in hand, and generally provides the better cognitive account of an expert's knowledge; the latter is a computational representation that can effectively solve the task in hand. Therefore, whatever language is used for developing a conceptual model, in the end it may not be isomorphic to the knowledge base. Typically, efficiency-related factors, which come into play at the knowledge encoding stage, heavily affect the design and refinement of the system. Moreover, emulating the expert may not be either viable or the best approach. For example, scheduling problems can often be solved by specialized algorithms, more efficiently than using the techniques employed by humans.

The dichotomy between the conceptualization and encoding phase has long formed the basis for a knowledge engineering methodology [11], and it is now widely accepted as the starting point for the construction of knowledge engineering environments. The main advantage of such a dichotomy is that it gives a clear-cut division between the problem of building a model of the problem-area, and that of encoding it onto a machine. However, for an effective set of guidelines to be provided both for the construction of knowledge-based systems and for the design of a knowledge engineering toolkit, these steps need to be analyzed in greater detail.

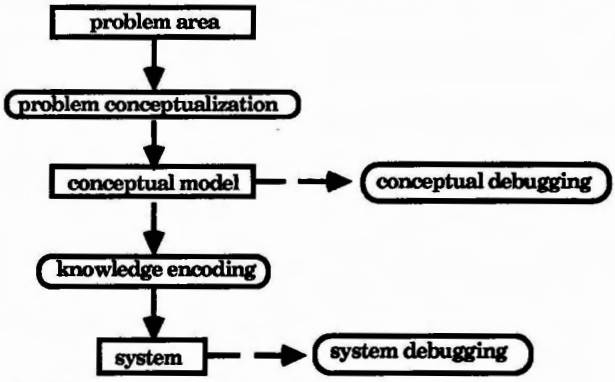



Figure 1. Broad description of knowledge engineering activities.

[1] Our notion of conceptual adequacy is basically the same as the notion of conceptual validity introduced in [40].

The processes of problem conceptualization, knowledge encoding, and debugging are discussed in sections 3, 4, and 5. Section 6 highlights the main points advocated in our discussion of the methodology, and indicates some future research directions.

## 3. Problem conceptualization

### *3.1. Knowledge acquisition vs problem conceptualization*

A conceptual model of a problem area is typically[1] produced through a process of knowledge acquisition. Knowledge acquisition efforts can be classified as either top-down or bottom-up.

Top-down knowledge acquisition is carried out to acquire new knowledge in relation to a pre-existing model, which is called the *skeletal model*. The top-down knowledge acquisition process is called *skeletal model instantiation*. The skeletal model determines the nature of the output produced by the knowledge acquisition activity. For instance, Aquinas [12] can elicit and conceptualize an expert's mental model by means of repertory grids. In this case, we can say that the acquisition of knowledge is driven by a psychological model, to produce a conceptual representation of human expertise. SALT [13], which is based on a propose-and-revise problem-solving model, and MOLE [14], which uses a heuristic classification model, also perform top-down knowledge acquisition. However, as the models they employ are based on the architecture of the emerging expert system, we say that they are driven by *design models*, which are abstract descriptions of the behaviour of a knowledge-based system, as discussed in section 4.1. A different situation is found in Teiresias [15], which performs a form of top-down knowledge acquisition based on rule models. In this case we say that the transfer of expertise happens at the *knowledge representation level* (see section 4.2).

The scope of the skeletal models can also vary considerably. For instance, a problem solving technique can be applicable in different domains and tasks, or alternatively can be very domain-specific.

In contrast to top-down methods, bottom-up knowledge acquisition is characterized by the lack of a strong skeletal model that can drive the transfer of expertise. For instance, general-purpose systems that produce conceptual models from documents can be viewed as performing bottom-up knowledge acquisition.

From the discussion above it should be clear that knowledge acquisition is not a stage of our methodology, but a collection of techniques that can be used to perform transfer of expertise at different levels of the knowledge engineering process. In Keats we provide support for both bottom-up and top-down knowledge acquisition as discussed in the next two sections, which are a summary and a revision of the discussion reported in [3, 4]. As they are only included here for the sake of having a complete description of how the toolkit supports the Keats methodology, they may be skipped by a reader who is already familiar with the Keats approach to knowledge acquisition.

### *3.2. Bottom-up knowledge acquisition in Keats*

We characterize bottom-up knowledge acquisition as comprising three subtasks, which are *knowledge elicitation*, *data analysis*, and *data conceptualization* (see figure 2).

#### *3.2.1. Knowledge elicitation*

Knowledge elicitation is the process of extracting knowledge from an expert, to produce what we call the *raw data*. These data can exhibit a high or low degree of organization depending on the technique being used. Such techniques range from structured and unstructured interviews with the expert(s), to psychologically based methods such as laddered grid or card sorting. Their efficacy

[1] A conceptual model can also be produced by selecting it from a library, or through analogical inferences.

varies a lot, depending on i) the type of knowledge the knowledge engineer is trying to obtain, ii) the task in hand, and iii) the psychological traits of the expert [16]. In general, it is true that the richer the set of data a technique can extract, the more noisy they are. Therefore, more work is usually needed before the elicited data are suitable for encoding onto a machine.

Keats doesn't provide support for knowledge elicitation, as it takes as a starting point the raw data elicited from the expert.

*3.2.2. Data analysis*

Once a set of data has been elicited, the knowledge engineer attends to the data analysis activity with the aim of removing the noise and useless material from the raw data, by identifying and extracting atomic 'chunks' of knowledge in the form of concepts, statements, definitions or relations. The way data analysis is carried out heavily depends on the knowledge elicitation technique that has been used; it is relatively straightforward, when formal techniques are employed, but it can be problematic if the raw data are particularly unstructured. At this level "the representation is superficial and has not been integrated into a conceptual framework. The same knowledge of different experts may have to be represented differently, because they use different terminology, or because their knowledge is structured differently" [17]. Thus, the so called 'digested data' express isolated chunks of knowledge that the expert can recognize, while not yet providing a model of the problem. The best way we know of characterizing this process of isolating chunks of knowledge, through glossaries and free associations is viewing it as the construction of a *Knowledge Encyclopedia*. This metaphor is further substantiated by the fact that at this level of analysis concepts are identified and associated with the relevant raw data, which help to describe them, much like entries in an encyclopedia. Examples of this approach can be found in the Cognosys system [18], that produces "the general content models of a domain before the use of a knowledge acquisition tool".

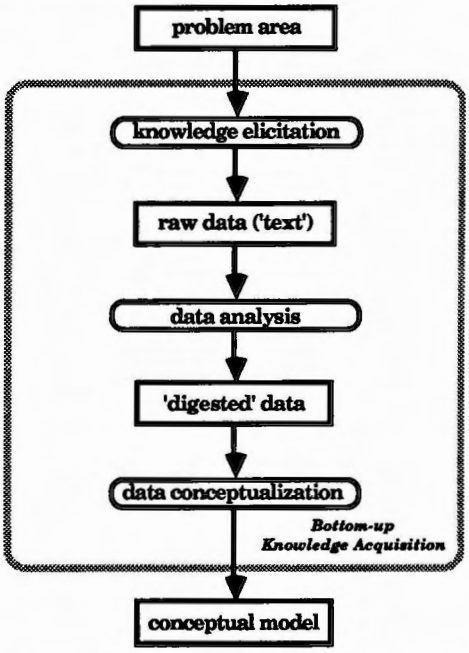



Figure 2. Bottom-up knowledge acquisition consists of knowledge elicitation, data analysis, and data conceptualization.

In Keats, the data analysis phase is supported by Acquist [5], a hypertext based facility [19] which allows the knowledge engineer to carry out data analysis and conceptualization through the process of abstracting and structuring the knowledge contained in documents. These documents can be transcripts from interviews, protocols of expert behaviour, books, etc. The chunks of text isolated during data analysis are called *fragments*. Fragments which refer to the same/similar piece of knowledge are clustered around a concept. A concept can be viewed either as an abstract label for a set of fragments, or a notion whose meaning is defined by the set of fragments. This implements the notion of the knowledge encyclopedia, where the concepts are entries, and the fragments are the description or definition of each entry.

*3.2.3. Data conceptualization*

At the data conceptualization stage, the knowledge engineer attempts to impose a global structure upon the data collected so far in order to produce a conceptual model of the problem.

Keats supports conceptualization with two additional mechanisms in Acquist. Firstly, concepts can be arranged in a hierarchy, allowing Acquist to deal with complex domains by partitioning the knowledge. Secondly, concepts can be linked together with user-defined relations enabling the knowledge engineer to construct conceptual models. To facilitate the development and visualization of conceptual networks, Keats also provides a graphical, WIMP based facility called *maps*. These provide the user with a CASE-like tool, as he or she can generate conceptual models by means of a freehand sketch of the problem area. The creation of a map from a set of related concepts can be also carried out automatically by Acquist for rapid viewing of emerging conceptual models. Figure 3 shows a screen layout of Acquist.

*3.3. Support for top-down knowledge acquisition in Keats*

In Keats we provide support for top-down knowledge acquisition by means of a Hypercard-based approach, organized around the creation of *coding sheets*. A coding sheet is a template defining the structure of a skeletal model, which can be used for driving the acquisition of knowledge (skeletal model instantiation). An example of a coding sheet used for a fault diagnosis interview session is given in figure 4.

The left-hand column of figure 4 is a place for noting the current symptom and where it comes from (e.g. 'fault docket'). We use the word 'clue' to indicate that diagnosis is not just symptom-based, but may be based on the outcome of previous intermediate results. Indeed, in our implementation, the 'source' entry includes an automatic (mouse-sensitive) link back to the coding sheet result which led us here in the first place. The second column can be used to indicate the particular diagnostic technique. The last four columns provide a means for recording any of six possible outcomes of a test, with the possibility of extending the sheet by pointing off to more results. Three standard interpretations are catered for ('win', 'lose', and 'don't know'), although these can be specified in more detail by the knowledge engineer. The new action is typically just a pointer off to more coding sheets, wherein the diagnostician follows through the initial interpretations.

This coding sheet is based on a model of expert behaviour which is organized around a *best-first* search paradigm [20]. This sheet proved effective in encoding most of the diagnostic behaviour of the expert (problem conceptualization). In fact it was also possible to quickly build a prototype whose architecture reflected the organization imposed by this coding sheet. However, the type of system so generated, organized around a decision tree diagnostic structure, was a 'shallow' one and, as such, of limited power. A much more powerful and principled implementation, as described in [2] could only be devised after developing functional and formal models of the domain and after an in-depth analysis of the expert's cognitive behaviour.

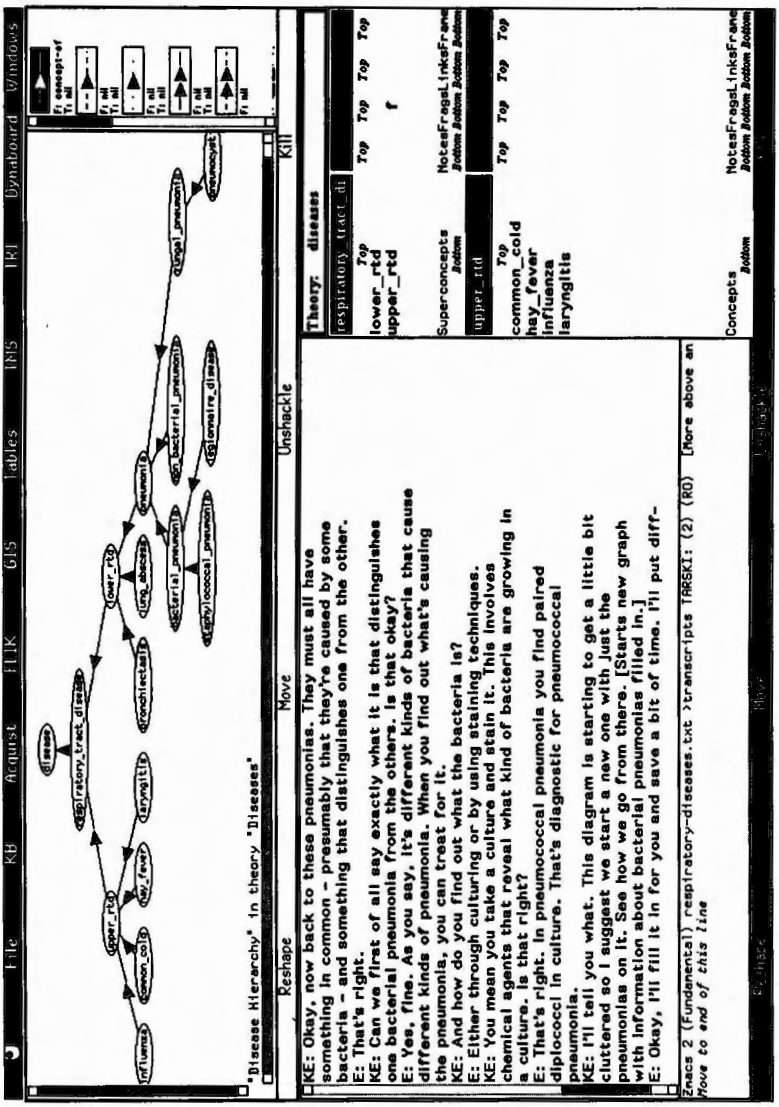



Figure 3. A screen snapshot of Acquist, showing the hypertext analysis tool in the lower half of the screen, and a map of the disease hierarchy in the top half. The scroll menus to the right of the editor contain a restricted view of the concept hierarchy and the attached fragments of text. The maps facility not only provides an overview of this hierarchy, but can also be used to show any user-defined relationship between the concepts. The command menu at the very top of the screen is used to navigate around the Keats toolkit.

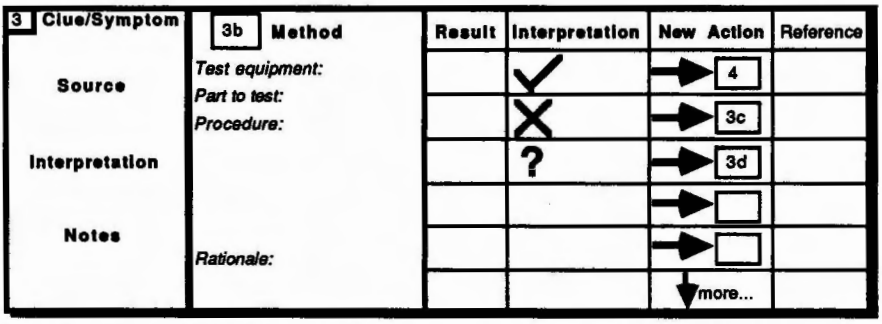



Figure 4. Coding sheet used for real-time encoding of expert behaviour on a fault diagnosis task.

In the neurobiology application described in [4] we developed a coding sheet that characterizes the 'domain phenomenology' of a particular class of scientific experiments. This approach proved very useful, not only because the sheet could be used for conducting top-down knowledge acquisition, but also because it provided a focal point for discussion between domain experts. For example, speculation about the outcome of future experiments could be expressed directly in terms of new symbols placed onto an existing coding sheet. Moreover, the complexities inherent in dealing with several foci of interest at once (behavioural, biochemical, physiological, and morphological), became manageable by virtue of being both visualizable and directly manipulable. The experts could use the coding sheet not only to make coherent statements about what they expect to be happening where and when, but also to point out specific misconceptions in the knowledge engineer's own evolving domain conceptualization. Indeed, the design of the coding sheet itself is typically an evolutionary process, during which the expert can highlight which items are salient, which attributes are worthy of having their own dimension, and which deserve a more fine-grained treatment.

### *3.4. Issues in problem conceptualization*

The methodological dichotomy between models of the problem area and models of the systems is reflected in the Keats architecture, which allows the generation of conceptual models to be a separate activity from the construction of the knowledge base.

Conceptual models are represented in Keats by means of a network of concepts and relations, developed using Acquist maps. Although other researchers [21, 22] have advocated the use of more sophisticated languages for describing conceptual models, we believe that particularly powerful languages are not required, as the abstract, computationally independent nature of conceptual models imposes little representational demands on a particular formalism.

More important than the particular modelling language is, we believe, the analysis of the structure of a conceptual model. In the KADS methodology [17, 23] the authors suggest a four layer architecture, describing the domain, the type of inferences, the task, and the strategic level. They also describe a number of domain independent, conceptual primitives used for representing the inference and task layers. Although they can't yet claim that such a set of primitives is complete, their proposal is already robust and detailed enough to be adopted by knowledge engineers. The main difference between our notion of conceptual models and KADS' interpretation models seems to concern the relationship between these models and the architecture of the end system. In our approach the two aspects are, at least in principle, kept separate. In the KADS approach interpretation models also provide "the architecture of a prospective expert system" [17]. Given this perspective they cannot truly be regarded as 'pure' conceptual models, as for example those based on psychological techniques. It is not yet clear what the relative trade-offs of these two approaches are. In the two examples mentioned in section 3.3 the conceptual models that we produced were in the end very different from the architectures of the end systems.

Nonetheless, in both examples, the coding sheets were conceptually adequate representations of the problem areas in hand, and, as such, they proved invaluable to help us to understand the domain. On the other hand, to explicitly produce conceptual models whose structure is different from the final architecture of the system can be too time-consuming. It is also possible to envisage situations in which the conceptual models are a perfectly adequate representation and should therefore be operationalized (i.e. mapped to structures in the knowledge base). In Keats, we let the knowledge engineer decide whether to keep conceptual models separate from the architecture of a knowledge-based system, as he or she can at any time operationalize a conceptual model, by mapping it to a knowledge representation language. Keats provides semi-automated support for perfoming such a mapping.

## 4. Knowledge encoding

Figure 5 shows a decomposition of the knowledge encoding task in terms of *design*, *knowledge representation*, and *implementation*. These subtasks are described in the next sections.

### *4.1. Design*

Design is the process of producing an abstract, machine and representation independent description of a knowledge base, and aims to identify and specify what has been called the *epistemological* [24] or *knowledge level* [25] description of a system. A knowledge level description of a system identifies its abstract data and inference types, and its generic control structure. For instance, in [26] a knowledge level account of the heuristic classification method of problem solving is given in terms of data, hypotheses, abstraction processes, refinement processes, and heuristic steps. Abstraction processes are used to categorize the data, refinement processes search for hypotheses that are more specific than those currently believed, and heuristic steps perform the mapping from data to hypotheses. As stressed in [25], an analysis at this level only describes the *competence* of the intended system, and says nothing about the processing aspects of the model, and the computational resources that it may need. For instance, the aforementioned model of heuristic classification could be encoded in a rule or in a frame-based system, using forward or backward chaining control.

This level of analysis is similar to the conceptual one, as it is independent from computational or representational issues. The main difference between the two levels is that, at the conceptual stage, the emphasis is on describing an external world, while at the design stage it focusses on the system that the knowledge engineer intends to develop. We believe that the explicit separation of the two levels makes it easier to split the conceptually-based problems from those associated with the abstract architecture of the intended expert system.

Unfortunately, there are no established guidelines on how to move from a conceptual model to a knowledge level description of a knowledge base. Automated, model-based knowledge acquisition systems, such as SALT or MOLE, overcome this problem by skipping the explicit production of a conceptual model, and focussing on producing a system based on a fixed problem solving technique, while knowledge engineers usually rely on empirical guidelines for associating problem solving techniques to tasks.

Support for the design task is generally provided by means of libraries of problem-solving methods and tasks, and heuristic guidance for matching the former to the latter [27, 28]. For example, constructive tasks can be tackled by means of propose-and-revise, or skeletal plan refinement strategies, while analysis tasks, in which the solution can be selected from a predefined set, can be solved using heuristic classification.

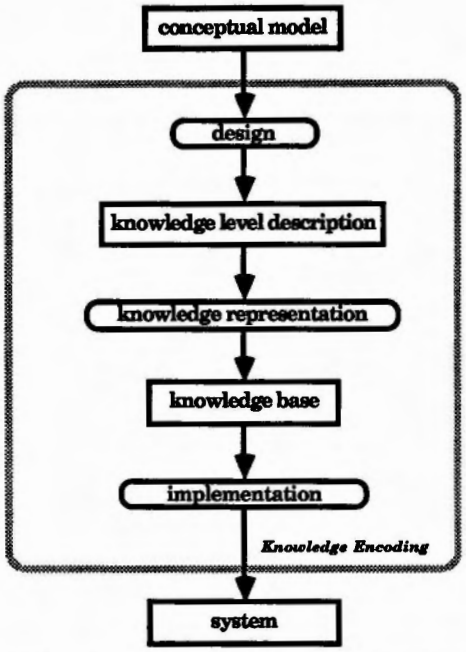



Figure 5. Knowledge encoding consists of design, knowledge representation, and implementation.

The choice of a language for describing the knowledge level of a system has also been debated. Advocates of logic have pointed out the advantages of formal analyses of the competence of a system [29, 30], but it is contentious whether it is actually necessary to have a formal approach to the problem [25]. Finally, other researchers [31] advocate the use of languages which can describe three aspects of a knowledge level analysis: the function of the model, the representation and organization of knowledge, and the control strategy.

In [32], the authors point out that the design process could be driven by the selection of an AI paradigm. Examples of AI paradigms are production systems and state-space, which provide ways of approaching the construction of a system. In the paper the authors speculate that more work on formulating AI paradigms will provide a basis for an understanding of the design process. In [33], a framework for evaluating knowledge acquisition methodologies, based on the concept of 'task-method' is sketched. The author proposes bridging the gap between task demands and problem solving methods, by operationalizing the tasks, that is by defining them in terms that include the problem solving techniques. The proposal, although rather abstract, is an interesting attempt at overcoming the gap between tasks and methods by providing a unifying framework.

Keats provides only weak support for the design phase, as there is no tool that is specifically aimed at supporting this task. However, coding sheets can be used to perform knowledge acquisition based on design models, while the facility for mapping conceptual models to knowledge bases can also be seen as support for the design phase.

### *4.2. Knowledge representation*

At the knowledge representation stage, the abstract architectures specified at the conceptual and design stages are made concrete, as knowledge representation formalisms and control regimes are selected. Although there aren't precise guidelines for deciding which formalism is the best in a particular case, there are some heuristics which are followed consistently by knowledge engineers. For instance, frames are normally used for representing taxonomical information, and rules for heuristic associations among facts in the domain. Backward chaining is a control regime

developed for performing goal-directed reasoning, while forward chaining rules can simulate data driven processing. A good discussion and a methodology for choosing representations given a task model can be found in [34]. The work that is being conducted on the CYC project [35] also outlines a methodology for choosing representations.

When we had to decide what kind of knowledge representation support to implement in Keats, we were faced with a trade-off between the simplicity provided by homogeneous languages and the power, flexibility, and efficiency provided by hybrid approaches. Simple rule-based shells, as well as more sophisticated, logic-based toolkits, such as Omega [36, 37], are examples of homogeneous approaches to knowledge representation. Typical commercial toolkits such as KEE™ are examples of the hybrid approach. In the end we decided to go for the hybrid solution, as our aim was to provide good overall support for skilled knowledge engineers, rather than neat but inflexible and computationally expensive languages. The knowledge representation formalisms provided by Keats are:

- ERI: a forward and backward chaining rule interpreter
- TMS: a non-monotonic truth maintenance system
- FLIK: a frame-based representation language
- CS: a constraint-based language

Due to space limitations, we won't describe them here. Instead, we'll comment on the methodological guidelines that have influenced the design of these languages.

In our experience, the main problem faced by users of hybrid toolkits is that of visualizing and debugging the behaviour of heteregeneous code, as knowledge engineers have to cope with different styles of representations, and different control regimes. To alleviate this problem, we have equipped the Keats toolkit with the visualization tools which are listed below.

- TMV: the Truth Maintenance Viewer
- GIS: a Graphical Interface System, which supports the visualization of a FLIK-based knowledge base
- TRI: the Transparent Rule Interpreter, which supports the visualization and debugging of rule-based models.
- TABLES: a spreadsheet-like interface, which, under user control, can be used to visualize any Keats knowledge representation formalism.

Moreover, to further facilitate the development and the debugging of hybrid knowledge bases, the Keats knowledge representation support consists of *integrated*, *clear*, and *flexible* formalisms, as discussed in the next three sections. The Keats approach to program visualization is briefly outlined in section 5.3.2, and, more in detail, in [8].

### *4.2.1. Integrated formalisms*

By integrated formalisms, we mean knowledge representation languages that either share the same structure or that can communicate through simple protocols. In Keats, all the knowledge representation formalisms map to a simple, *functional* architecture organized in terms of *facts* and *knowledge sources*. For instance, the fact "bill is the father of mary" can be internally encoded in any of the four representations that Keats provides, and externally accessed in a way that is independent of the formalism. The same applies to the inferences that are carried out by the system. While new facts can be deduced through inheritance, rule application, or constraint propagation, at the functional level, we are only concerned with the link (knowledge source) that connects the newly deduced facts with those that have made the inference possible.

The provision of integrated formalisms greatly facilitates both visualization and debugging. In fact, at a coarse-grained level, a single visualization tool can be applied to view heterogeneous formalisms, as long as their functional descriptions are homogeneous. Analogously, when debugging a program, the knowledge engineer can first examine the functional description of the knowledge base, and later inspect the specific structures, inference or control mechanisms, which

have caused the bug. In Keats, we use the TMS, and its associated visualization tool (TMV) [8], to provide a uniform view of a knowledge base. In fact, every fact and inference in the knowledge base can be associated with a TMS node or justification, which provides a uniform description of a heterogeneous knowledge base.

*4.2.2. Clear execution models*

In order to support this integration principle, we need to have formalisms characterized by a clean and clear execution model. For instance, in order for inheritance-based inferences to be describable in terms of the TMS, the FLIK inheritance mechanism has to be sound. It is worth noting that this property is not attained by most frame based representations [38, 39]. Analogously, in designing the constraint language, we have opted for a simple but clean forward-propagation model.

*4.2.3. Flexible control regimes*

Finally, in order to cater for the needs of experienced knowledge engineers, knowledge representation formalisms should be flexible. That is they should provide control regimes that can be modified and enhanced by the user. Each of the Keats knowledge representation formalisms allows the user partially to extend it. For instance, contradiction handlers can be defined by the TMS user, FLIK provides metalevel capabilities and mechanisms for tuning the inheritance, and ERI provides user-definable conflict resolution strategies.

*4.3. Implementation*

At the implementation stage, decisions about the hardware, the integration with other software modules, the documentation and packaging of the system have to be taken. We will not discuss these issues here and we refer the interested reader to [9].

## 5. Debugging and maintenance

The knowledge engineering methodology proposed in this paper is based on the successive refinement of qualitatively different models. Due to this sequential process, any bug, which is introduced in a particular model, will be inherited by successive ones, where it may manifest itself. Therefore, debugging necessitates tracing back from the bug's manifestation, through the models, to its point of introduction. For example, errors may be introduced during the knowledge acquisition stage, as the knowledge engineer (or the expert) can make mistakes during the elicitation and analysis phases. A buggy conceptual model will inevitably produce a buggy knowledge base. In other cases a correct conceptual model can be erroneously formalized into a knowledge base. For instance, in rule-based systems, the incorrect behaviour of a rulebase may be the consequence either of a buggy rule or of the incorrect characterization of the conceptual model of the expert's reasoning. In the former case the knowledge engineer needs only to locate the buggy rule. However, in the latter case the rule is a correct implementation of a buggy part of the conceptual model and it is therefore necessary not only to find the rule but to also locate the buggy component of the conceptual model. Thus, we require debugging to be *both an intra and inter model process*. In Keats we support inter model debugging by providing a coarse 'history' for each model component throughout the model refinement process, allowing the user to inspect the development of a component from 'raw data' to 'system'. For example, each frame in the knowledge base has a direct link to the corresponding concept in the conceptual model, which has links to the pertinent raw data. This feature corresponds to the integration principle advocated in section 1.

Determining the quality of a model with respect to its predecessor requires *validation* and *verification* of the information that has been transformed. In [40] the authors define validation as "the degree of homomorphism between the representation system and the system it is supposed to

represent [41]", while in [42] verification is defined as "the demonstration of consistency, completeness, and correctness of the software at each stage and between each stage of the software development life-cycle". Our view of knowledge engineering as the production of a number of qualitatively different but interrelated models implies that different aspects are important at different stages of the knowledge engineering process. For instance, we shall see that *validation* is meaningful at the conceptual level, while *verification* is important at the knowledge representation level. Maintainability is also a very important factor at the knowledge representation level, as it gives an estimate of how easy it is to debug and extend a knowledge base.

From a methodological point of view it is important to devise guidelines that could help the user to develop knowledge bases that score high on the criteria listed above.

### *5.1. Validation*

Validation assesses the mapping between a source and a target system. In knowledge engineering, validity should provide a measure of how faithful the system's architecture and behaviour are with respect to the problem area. Unfortunately, this comparison is made difficult because of the many confounding factors that can arise at the implementation level, such as efficiency, software design and machine oriented issues. Therefore, we believe validity should mainly be concerned with assessing the adequacy of the conceptual model developed. At the knowledge encoding level, validity becomes a fuzzier concept, as computational issues may lead the knowledge engineer to design system architectures that are structurally different from the conceptual model.

In [40] a number of methods for assessing the validity of conceptual models are suggested.

### *5.2. Verification*

Verification is the process of assessing the consistency, correctness, and completeness of a knowledge base. Various techniques have been suggested for performing this task [43, 44, 45, 46]. Keats doesn't yet provide any software support for verifying a knowledge base, although we envisage that some of the aforementioned techniques could easily be embedded in the toolkit.

From a methodological point of view the important aspect is that the knowledge base should be formalized using semantically sound and syntactically simple formalisms (such as logic or production systems), which can support formal methods for verifying the knowledge base. For instance, frame based systems relying on arbitrary forms of inheritance cannot be formally analyzed [38]. While correctness and consistency checking normally depend on the formalism being used, completeness is very much a semantic property depending upon the task or the problem solving method being employed. For instance, SALT uses a definition of completeness that is based on its propose-and-revise problem solving model. Again, in order to build knowledge bases for which completeness is a definable notion, it is important that the problem solving and task models are made explicit, as prescribed by our methodology.

### *5.3. Maintainability*

The maintainability of a knowledge base is a measure of how easy is to extend and debug it. As discussed at the beginning of section 5, debugging requires the programmer to locate the manifestation of a bug and its root cause. Clearly, the two components of maintenance are related, as the easier it is to extend a knowledge base, the easier it will be to debug it. For the purposes of exposition we will discuss the two aspects separately.

#### *5.3.1. Facilitating the extension of knowledge bases*

There are many different aspects of a knowledge base that affect its ease of extension. In [47] two such aspects are described, *homogeneity* and *predictability*. Homogeneous code uses a small number of stereotypic language constructs, called *plans* [48, 49, 50] to achieve desired goals. Predictable code is defined as having the following three properties: i) users know where to look

for answers to questions, ii) users are not surprised by what they find, and iii) nothing untoward is carried out behind the scenes. Another aspect affecting ease of extension is *modularity*. A system is said to be modular if the interactions between components are minimized and they use well specified protocols. This notion comes from traditional software engineering methodology and has long been advocated in AI programming as well [51]. Production systems provide an example of a modular programming formalism, as interactions between procedures (rules) are forced through a very narrow channel (working memory). Further modularity can be achieved by clustering rules into groups, in order to limit the number of possible interactions. In [52], the authors describe a method for automatically partitioning a rulebase into groups in a way that achieves a high degree of modularity. Exceptional predicates that occur in more than one partition need to be documented by the programmer.

In Keats each knowledge source has an explicit representation of its functional interface to other knowledge sources, by means of *contract predicates*. Contract predicates are the subset of predicates within a knowledge source that also occur in any other knowledge source, and are thus the channel of communication between them. Contract predicates can be used in two ways. Firstly, when augmenting a knowledge source a programmer will know that only contract predicates will interact with the rest of the knowledge base. Secondly, a programmer can, if desired, document each contract predicate specifying how exactly it should interact with the rest of the knowledge base.

### *5.3.2. Locating bugs in knowledge bases*

*5.3.2.1. Using different levels of abstraction to facilitate debugging.* Although, as described in section 5.2, various techniques have been specified for the static verification of a knowledge base, less work seems to have been carried out in providing knowledge base debugging facilities. One notable exception is the Oregon Rule Based System (ORBS) [53], a rapid prototyping environment for experienced rule based programmers. ORBS relies on an Interlisp style break package, which allows programmers to insert breaks into their rulebase. ORBS also provides tools such as a micro matcher which displays a low level account of the generation of a rule instantiation.

Within Keats we have taken a different approach. It is our conjecture that, given a clear visualization of program behaviour, the task of bug location can be much simplified. For instance, there are two ways one might wish to view a rulebase in order to debug it. In a *coarse-grained* (global) view, the user requires some idea of the overall pattern of execution of the rules in the program. From a *fine-grained* (low-level) point of view, detailed information about unification, rule instantiations, and the state of working memory is needed. Both of these views are required and if either are absent one gets one of two difficulties occurring. First, without a global view of execution it is very difficult to pinpoint the part of the execution space in which a bug has occurred. Second, without a fine-grained view of execution, it is impossible to answer low-level questions such as why a certain rule did not fire in a certain cycle. Several existing knowledge engineering environments currently available, such as KEE™ and NEXPERT™ provide rule execution traces, but, because they use inappropriate abstractions, they do not provide a true coarse-grained view [7]. Therefore, in Keats we provide visualization tools that present multiple views of the representation at varying degrees of abstraction.

As an example of our approach we show how TRI allows the user to easily visualize and debug rule-based systems.

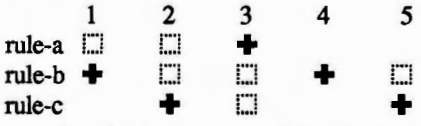



Figure 6. A graph showing the execution of the rules *rule-a, rule-b* and *rule-c* over five cycles.

*5.3.2.2. Debugging with TRI.* TRI represents the coarse-grained behaviour of a forward-chaining rule interpreter by providing an explicit representation of the time dimension. A simple example is shown in figure 6.

The symbols in figure 6 denote the following. A ✚ means that a rule fired, a ⬚ means that a rule entered the conflict resolution set but did not fire. We can see from figure 6 that the rules fired in the order: *rule-b, rule-c, rule-a, rule-b, rule-c* and that *rule-a* entered the conflict resolution set during the first 3 cycles only.

Given the style of display shown in figure 6, the user can easily visualize an incorrect behaviour, such as a rule firing unexpectedly, and then examine in detail (fine-grained view) the state of the environment at the moment the incorrect behaviour was recorded.

The issue of what to display at a fine-grained level is much simpler (e.g. variable bindings, working memory contents), and in many ways has been addressed already by generations of textual-based tracers for production systems such as OPS5 [54]. However, TRI provides two novel extensions to earlier (textual-based) fine-grained displays.

- *High selectivity.* The user can select elements for fine-grained viewing in a range of ways, depending upon the time, the rule, the working memory pattern, or any combination of these.
- *Synchronization with coarse-grained view.* Both views can be presented simultaneously, so that as the coarse-grained view is manipulated (e.g. 'played forward'), the fine-grained view changes appropriately.

For example, a slice of the conflict resolution set can be chosen by clicking on one of the nodes in the graph. Figure 7 shows a fine-grained view of the ✚ node for *rule-a* at cycle 3.

The frame in figure 7 has two components. The top part of the frame contains the definition of the rule; the bottom part shows all the instantiations the rule had in the conflict set. This display allows easy comparison of the instantiated rule with the source code. The first instantiation is in italics indicating that this instantiation fired in cycle 3. Each instantiation and each clause within each instantiation are mouse sensitive, allowing operations such as displaying the deleting conflict resolution strategy (for an instantiation) or highlighting the 'depositing' rule in the graph (for a clause).

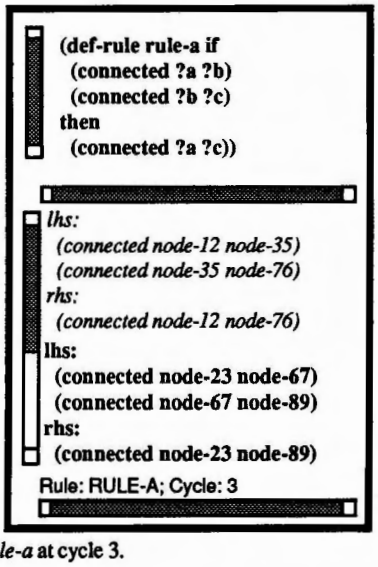



Figure 7. A fine-grained view for *rule-a* at cycle 3.

## 6. Conclusions

To summarize the important ideas inherent in the Keats methodology, we have formulated what we loosely term the 'Keats manifesto'. It is not meant to be prescriptive as such, but rather to highlight the key themes which permeate the Keats software and methodology.

*Keats manifesto*

- *Primacy of functional stages over technology.* The Keats approach emphasises an understanding of the nature and functional decomposition of knowledge engineering, rather than its technological, organizational and environmental aspects.
- *Representation independence.* Our analysis holds independently of particular knowledge acquisition or knowledge representation techniques.
- *KE = Model Refinement.* The transformation from some unknown representation in the expert's head to a formal representation resident inside the machine is a process of 'model refinement', where each of the knowledge engineering stages produces a qualitatively different model of what an expert knows about a domain.
- *KE = Problem conceptualization, knowledge encoding, debugging.* At a coarse level of description there is a useful, functional distinction between the activities associated with understanding the problem in hand, and those related to its formalization and encoding onto a machine. Both these tasks require debugging guidelines and tools.
- *Encoding = design + representation + implementation.* During knowledge encoding, it is useful to distinguish between the abstract architecture of the expert system, the knowledge representation techniques being employed, and the remaining software issues.
- *Knowledge acquisition is a collection of techniques.* Knowledge acquisition is not a stage of our methodology, but a collection of techniques that can be used to perform transfer of expertise at different levels of the knowledge engineering process.
- *KE toolkit should provide principled, integrated, and comprehensive support.* A knowledge engineering toolkit should provide integrated support for the various stages specified by its associated methodology.
- *Formalism integration.* Knowledge representation languages should either share the same structure or be able to communicate through simple protocols. In Keats, all the knowledge representation formalisms map to a simple, *functional* architecture organized in terms of *facts* and *knowledge sources*.
- *Formalism transparency.* Knowledge representation languages should have a clear execution model.
- *Formalism flexibility.* Users should be able to extend knowledge representation formalisms by implementing their own forms of control regimes, provided soundness is maintained.
- *Debugging is both an intra and inter model process.* In Keats we support inter model debugging by providing a history for each model component throughout the model refinement process, allowing the user to inspect the development of a component from 'raw data' to 'system'.
- *Validation at the conceptual level, verification at the knowledge representation level.* Because confounding implementational issues come into play at the knowledge representation stage, validation is a notion associated with the conceptual level. Verification requires the use of semantically sound and syntactically simple

knowledge representation formalisms. Explicit task and problem-solving models facilitate the definition of problem-dependent notions of completeness.

- *Clear visualization greatly simplifies debugging.* Providing a clear visualization of program behaviour enables the causes of bugs to be pinpointed rapidly.

The status of Keats as both a methodology and a software development environment has always been a source of potential confusion. As this paper has argued, however, the two really go hand in hand, and one cannot satisfactorily work in one frame of reference without due regard for the other. Knowledge acquisition tools which don't tie in to an ultimate implementation pathway are frustrating, as are representation and implementation vehicles which pay no attention to the all-important phase of problem conceptualization. Our aim has therefore been to link methodology and software tools, and to do so in a manner which relates to all phases of the knowledge engineering life-cycle. As mentioned at the beginning of the paper, our ideal toolkit should provide principled, integrated, and comprehensive support. Keats provides principled and integrated support, but doesn't yet cover all the stages of our knowledge engineering model. Moreover, this model is only a subset of what a complete, multi-players life-cycle model of knowledge engineering should be. Keats is the most comprehensive attempt we know of in this direction, yet it only scratches the surface of methodological and software life-cycle support issues!

## References


[1] Rappaport, A. T., and Gaines, B. R. Integration of Acquisition and Performance. *Proceedings of the 3rd Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, 1988.

[2] Motta, E., Eisenstadt, M., Pitman K., and West M. Support for Knowledge Acquisition in the Knowledge Engineer's Assistant (KEATS). *Expert Systems, 5(1)*, 1988, pp. 6-28.

[3] Motta, E., Rajan, T., and Eisenstadt, M. A methodology and tool for knowledge acquisition in KEATS-2. A. Guida and T. Tasso (Eds.), *Topics in expert system design*, Amsterdam: Elsevier, 1989.

[4] Motta, E., Rajan, T., and Eisenstadt, M. Knowledge acquisition as a process of model refinement. *Knowledge Acquisition*, 2, 1990, pp. 21- 49.

[5] Rajan, T., Motta, E., and Eisenstadt, M. Acquist: A Tool for Knowledge Acquisition. B. Kelly and A. Rector (Eds.), *Research and Development in Expert Systems V*, Cambridge University Press, Cambridge, UK, 1988, pp. 113-125.

[6] Domingue, J. TRI: The Transparent Rule Interpreter. B. Kelly and A. Rector (Eds.), *Research and Development in Expert Systems V*, Cambridge University Press, Cambridge, UK, 1988, pp. 126-138.

[7] Domingue, J., and Eisenstadt, M. A new metaphor for the graphical tracing of forward chaining rule execution. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*. San Mateo, CA: Morgan Kaufmann, 1989.

[8] Eisenstadt, M., Domingue, J., Rajan, T., and Motta, E. Visual Knowledge Engineering. *IEEE Transactions on Software Engineering*, In Press.

[9] Guida, G., and Tasso, C. Building expert systems: From life cycle to development methodology. A. Guida and T. Tasso (Eds.), *Topics in expert system design*, Amsterdam: Elsevier, 1989.

[10] Boehm, B. W. A Spiral Model of Software Development and Enhancement. *Computer*, 21, pp. 61-72, May 1988.

[11] Buchanan, B.G., Barstow, D., Bechtal, R., Bennet, J., Clancey, W., Kulikowski, C., Mitchell, T., and Waterman, D.A. Constructing an expert system. Hayes-Roth, F., Waterman, D.A., and Lenat, D.B (Eds), *Building Expert Systems*, Addison-Wesley, Reading, MA, pp. 127-167, 1983.

[12] Boose, J. H., and Bradshaw, J. M. Expertise transfer and complex problems: Using Aquinas as a knowledge acquisition workbench for knowledge-based systems. *International Journal of Man-Machine Studies, 26*, 1987.

[13] Marcus, S., and McDermott, J. SALT: A Knowledge Acquisition Language for Propose-and-Revise Systems. *Journal of Artificial Intelligence*, 39, 1989.

[14] Eshelman, L., and McDermott, J. MOLE: A Knowledge Acquisition Tool that uses its Head. *Proceedings of AAAI '86*, Philadelphia, PA, 1986.

[15] Davis, R. Interactive transfer of expertise: acquisition of new inference rules. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, Cambridge, Ma, 1977.

[16] Burton, A.M., Shadbolt, N.R., Hedgecock, A.P.and Rugg, G. A formal evaluation of knowledge elicitation techniques for expert systems: domain 1. D. S. Moralee (Ed.), *Research and Development in Expert Systems IV*, Cambridge University Press, 1987.

[17] Breuker, J., and Wielinga, B. KADS: Structured Knowledge Acquisition for Expert Systems. *Proceedings of Expert Systems and their Applications*, Avignon, 1985.

[18] Woodward, B. Knowledge Acquisition at the Front-End: Defining the Domain. *Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, November 1988.

[19] Conklin, J. Hypertext: An Introduction and Survey. *IEEE Computer Magazine*, pp. 17-41, 1987.

[20] Rappaport, A.T. Cognitive Primitives. International Journal of Man-Machine Studies, 29, 1988.

[21] Sowa, J. *Conceptual Structures: Information Processing in Mind and Machine*, Addison Wesley, 1984.

[22] Johnson, N.E., and Tomlinson, C.M. Knowledge Representation for Knowledge Elicitation. *Proceedings of the 3rd Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, 1988.

[23] Breuker, J., and Wielinga, B. Models of expertise in knoweldge acquisition. A. Guida and T. Tasso (Eds.), *Topics in expert system design*, Amsterdam: Elsevier, 1989.

[24] McCarthy, J., and Hayes, P. J. Some philosophical problems from the standpoint of artificial intelligence. Meltzer, B. and Michie D.(Eds.), *Machine Intelligence 4*, Edinburgh University Press, 1969.

[25] Newell, A. The Knowledge Level, *Journal of Artificial Intelligence*, 18. 1982.

[26] Clancey, W. Heuristic Classification. *Journal of Artificial Intelligence*, 27, 1985.

[27] Kitto, C.M., and Boose, J.H. Selecting knowledge acquisition tools and strategies based on application characteristics. *International Journal of Man-Machine Studies*, 31, 1989.

[28] McDermott, J. Preliminary steps toward a taxonomy of problem-solving methods. In Automating Knowledge Acquisition for Expert Systems, Marcus (ed), Kluwer Academic, 1988.

[29] Hayes, P.J. In Defence of Logic. *Proceedings of the fifth International Joint Conference on Artificial Intelligence (IJCAI '77)*, Cambridge, MA, 1977.

[30] Levesque, H J. A Knowledge-level Account of Abduction. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, (IJCAI '89)*, Detroit, 1989.

[31] Chandrasekaran, B. Towards a Functional Architecture for Intelligence Based on Generic Information Processing Tasks. *Proceedings of the International Joint Conference on Artificial Inteligence (IJCAI '87)*, Milan, 1987.

[32] Schreiber, G., Breuker, J., Bredeweg, B., and Wielinga, B. Modelling in KBS Development. *Proceedings of the Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems, EKAW '88*, Bonn, June, 1988.

[33] Woodward, B. Integrating Task Demands and Problem-Solving Methods to Develop Useful Taxonomies for Knowledge Acquisition. Proceedings of the Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems, EKAW '89. Paris, 1989.

[34] Reichgelt, H., and Van Harmelen, F. Criteria for choosing representation languages and control regimes for expert systems. *The knowledge engineering review 1(4)*, 1986.

[35] Lenat, D.B., and Guha, R.V. *Building Large Knowledge-Based Systems*. Addison-Wesley, Reading, MA, 1990.

[36] Attardi, G., and Simi, M. A Description Oriented Logic for Building Knowledge Bases, *Proceedings of the IEEE*, October 1986.

[37] Simi, M., and Motta, E. Amalgamation of Theory and Metatheory in Omega. Maes and Nardi (ed), *Meta level Architecture and Reflection*, North Holland, 1988.

[38] Nado R., and Fikes R. Semantically Sound Inheritance for a Formally Defined Frame Language with Defaults. *Proceedings of AAAI '87*, Seattle, Washington, 1987.

[39] Touretzky, D. S., *The Mathematics of Inheritance Systems*, Morgan Kaufmann, Los Altos, CA, 1986.

[40] Benbasat, I., and Dhaliwal, J.S. The Validation of Knowledge Acquisition: Methodologies and Techniques. *Proceedings of the Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems, EKAW '89*. Paris, 1989.

[41] Vandierendock, A. Inferential Simulation: Hypothesis-testing by Computer Simulation. *Nederlands Tijdschrift voor de Psychologie*, Vol. 30, pp. 677-700. 1975.

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

[43] Rousset, M-C. On the Consistency of Knowledge Bases: The COVADIS System. *Proceedings of European Conference on Artificial Intelligence, Munich*, August 1-5, 1988. pp. 79-84

[44] Perkins, W.A., Laffey, T.J., Pecora, D. and Nguyen, T.A. Knowledge base verification, in *Topics in Expert Systems Design, Methodologies and Tools* (Guida, G., and Tasso, C., eds), pp. 353-376, North Holland, Amsterdam, 1989.

[45] Reubenstein, H.B. OPMAN: An OPS5 Rule Base Editing and Maintenance Package. *Master's Thesis*. MIT Department of Electrical Engineering and Computer Science, June, 1985.

[46] Ayel, M. Protocols for Consistency Checking in Expert System Knowledge Bases. *Proceedings of European Conference on Artificial Intelligence*, Munich, August 1-5, 1988. pp. 220- 225.

[47] Soloway, E., Bachant, J. and Jensen, K. Assessing the Maintainability of XCON-in-RIME: Coping with the Problems of a VERY Large Rule-Base. Proceedings of the Sixth National Conference on Artificial Intelligence, Seattle, Washington, July 13-17, 1987. pp. 824-829.

[48] Rich, C. Inspection Methods in Programming. *Technical Report AI-TR-604*, MIT AI Lab, Cambridge, MA, 1981

[49] Soloway, E., and Ehrlich K. Empirical Studies of Programming Knowledge. *IEEE Transactions on Software Engineering*, SE-10(5), pp. 595-609, 1984

[50] Johnson, L.W., and Soloway, E. PROUST: An Automatic Debugger for Pascal Programs. *BYTE, The Small Systems Journal*. Vol. 10, No. 4., pp. 179-190, April 1985.

[51] Winograd, T. Frame Representations and the Declarative/Procedural Controversy. Bobrow, D. and Collins, A. (Eds), *Representation and Understanding, Studies in Cognitive Science*, Academic Press, New York, 1975

[52] Jacob, R.J.K., and Froscher, J.N. Facilitating Change in Rule-based Systems. In Hendler, J. A. (Ed) *Expert Systems: The User Interface*. Ablex Publishing Corp. Norwood, New Jersey, 1988.

[53] Fickas, S. Supporting the Programmer of a Rule Based Language. *Expert Systems*, Vol. 4, No. 2. pp. 74-87. May 1987.

[54] Brownston, L., Farrell, R., Kant, E., and Martin, N. Programming expert system in OPS5: an introduction to rule based programming. Reading, MA: Addison-Wesley, 1985.

# Decision Tree Induction using Domain Knowledge

Marlon Núñez
TELEFONICA I+D
C/ Emilio Vargas, 6
28043 Madrid, Spain
E-mail: mnunez@tid.ts.es

**Abstract.** At present, decision trees generated by algorithms of the ID3 family are accurate, however they can be incomprehensible to experts. There have been proposed various solutions to this problem: Cendrowska's PRISM and Gaines's INDUCT concentrates on finding the relevant values of attributes (by-passing trees), Quinlan (1989) concentrates on the relevant attributes overall, etc. The purpose of this paper is to show that the cause of this incomprehensibility is not only present within the historical data, but also outside this data: in the domain knowledge. The EG2 algorithm (Núñez 1988a, 1988b) presented in this paper tries to generate more logical and convenient decision trees applying various types of generalization and at the same time optimizing the classification cost by means of the use of domain knowledge. The domain knowledge contains the ISA hierarchy and the measurement cost associated to each attribute. The user can define the degrees of economy and generalization. These data will influence directly on quantity of search that the algorithm must undertake. This algorithm has been implemented and its results appear in this paper comparing them with other methods.

## 1. Introduction

The algorithms of the ID3 family have suffered various limitations with respect to its utilization in many kind of problems. Although the decision trees generated by these methods are accurate, they suffer the disadvantage of excessive complexity and are therefore incomprehensible to experts.

In order to solve the above problem, many systems have been developed. These systems concentrate on the information contained in the historical data: PRISM algorithm (Cendrowska, 1989) and INDUCT algorithm (Gaines, 1989) maximize the information contributed by an attribute-value pair to know a particular classification. Previously Quinlan (1989,1986), Lavrac *et al.* (1986), Kononenko *et al.* (1984), Breiman *et al.* (1984) proposed other techniques that were applyed on the historical data to generate more understandable and accurate decision trees.

In order to solve this problem, this paper will show that the cause of this incomprehensibility is not only present within the historical data, but also outside this data: in the domain knowledge. The lack of the domain knowledge limits the process of learning. One of these limitations consists of the impossibility of applying different types of generalization. In inductive algorithms that use languages based on logic of predicates like AQ11 (Michalski, 1983), INDUCE (Larson and Michalski, 1977) this function is basic and inseparable from the inductive process itself.

On the other hand, one of the main difficulties appears with diagnosis problems that include costly tests. One needs to acquire knowledge that economizes resources, that is to say, the learner should perform economic induction.

When an expert in brain tumours, for instance, receives a patient who suffers a headache, he does not recommend the Scanner as a first diagnostic test, this being the most effective one, because the expert has in mind the economic criteria. Therefore the expert asks simple questions and orders other more economic tests in order to discriminate the simple cases, and only recommends such an expensive test for the complex ones. This means that learning methods should take into consideration minimizing costs of the diagnosis in the process of rule induction, in order to be applicable to more problems.

The algorithm presented in this paper undertakes various types of generalization and optimizes the classification cost by means of domain knowledge. This algorithm, called EG2 (Economic Generalizer 2) (Nuñez, 1988a, 1988b) constitutes the base of ALEXIS II program whose results appear in this paper. Although this program represents instances in terms of attribute-value pairs, the EG2 algorithm can handle composite objects as training examples. However, this facility is being implemented at the moment and therefore has not yet been proved with real problems. For this reason it is not presented in this paper.

The following figure illustrates the role of EG2 algorithm as a knowledge acquisition technique using the same framework presented by Gaines (1989) in which we emphasise in the necessity of acquiring domain knowledge in order to use it in the induction process.

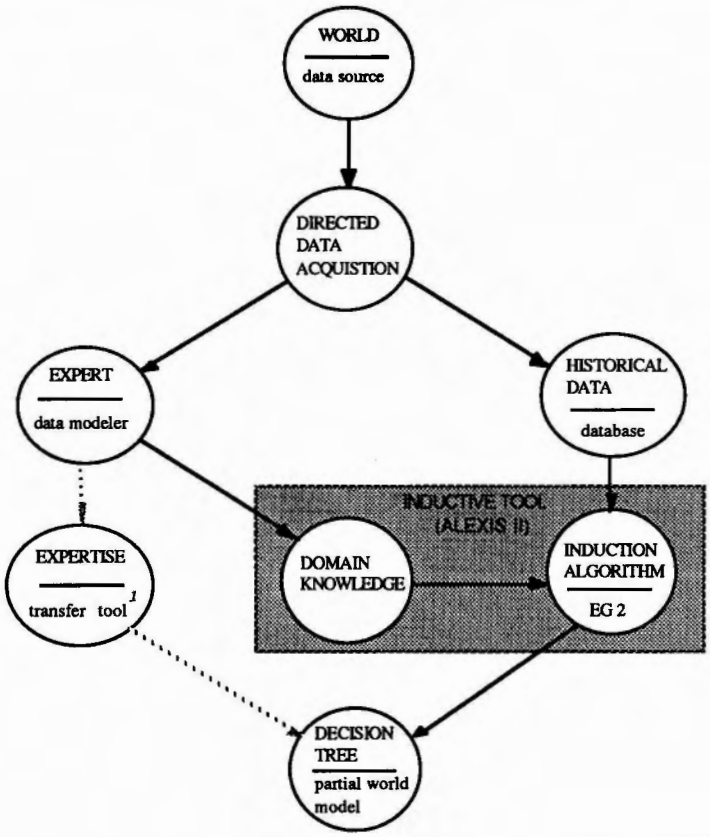



Fig 1 . Illustration of the role of EG2 induction algorithm in knowledge acquisition .

## 2. Representation and Organization in EG2

The EG2 algorithm is an inductive algorithm that generates a decision tree from a set of examples. Beside the training examples, the user can define the ISA hierarchy and the cost of measurement of each attribute, and some data about the degrees of economy and generalization. These data will influence directly on quantity of search that the algorithm must undertake.

---

[1] Although ALEXIS II induction tool has been used for learning from a database, we are working on numeric attributes to make ALEXIS II learn decision trees from information contained in grids.

### *2.1. Economic Induction*

With the goal of constructing the decision tree, the algorithm has a criterion of selection of attributes based on a relationship "cost/benefit". This criterion is more general than the one proposed by Quinlan (1979), based exclusively on the measurement of the quantity of information ($\Delta I$). EG2 will produce a decision tree which will allow the final user to classify his problem and in turn, economize on his resources: time, money, labour, energy, level of danger, etc. The average cost of classification with the decision tree generated by EG2 is less or equal to that obtained using the tree generated by ID3.

The economic criterion applied by EG2 to build a simple decision tree is based on the selection of the attribute with the least ICF (Information Cost Function). This function is defined as the ratio between cost and the information efficiency of the attribute, that is to say, its relationship "cost/benefit". The demonstration of how this formula has been obtained is fully documented in (Núñez 1988b). The ICF formula is as follows:

$$ICF_a = \frac{(cost_a + 1)^{\omega}}{2^{\Delta I_a} - 1}$$

where:

$\Delta I_a$ is the information gain of attribute a
$\omega$ is a variable of calibration of this economic criterion

The cost concept should be understood, not only as a monetary unit to measure a value of an attribute, but also as a concept that can be quantified i.e. distance, time, risk, danger level, etc.

ICF includes a variable $\omega$ to calibrate the factor of economy. The range of $\omega$ is [0,1]: $\omega = 1$ means maximum of economy. ICF is calculated for each attribute. EG2 selects the attribute with the least ICF.

### *2.2 Kinds of generalization*

#### *2.2.1 Climbing the generalization tree*

This kind of generalization has been used in various forms (Kodratoff, 1986; Michalski, 1983; Winston, 1977), and because of its relevance to concept recognition it has been included in EG2. The algorithm generalizes depending on the ISA hierarchies of values associated to the attributes. For an ISA relation to be acceptable in a decision tree it has to fulfil certain criteria of completeness and consistency.

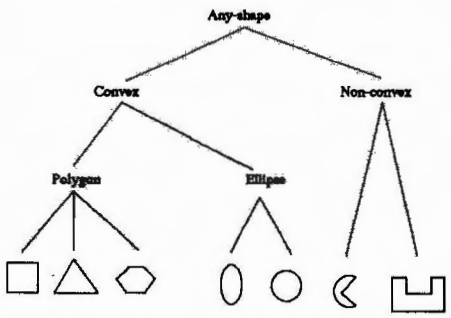



Fig. 2. An ISA hierarchy of values associated to an attribute.

*2.2.2 Union of values*

The algorithm can apply the union of symbolic values. If a union fulfils certain criteria of consistency and completeness the system accepts this union. In general terms, the specification input/output of the EG2 algorithm is given in the following figure:

---

GIVEN:

- DOMAIN KNOWLEDGE
  - . ISA hierarchies
  - . Measuring cost
- USER DEFINED FACTORS
  - . Economic factor $\omega$ (0..1)
  - . Completeness threshold ct (0..1)
- TRAINING INSTANCES:
  - Training examples

DETERMINE

- A decision tree

---

Fig. 3. Specification of the EG2 algorithm

## 3. EG2 Algorithm

Before describing the algorithm, we shall define as structured attributes those that have an ISA hierarchy associated to it (see figure 2). The lowest leaf values of this tree are the only observable values. The other values are called abstract values. The hierarchies can be "tangled" (Kodratoff et al, 1986), that is to say, an observable value can belong to more than one abstract value.

*3.1. Description of the Algorithm*

EG2 is a generalization of the ID3 algorithm. The algorithm works as follows: EG2 selects the attributes using the economic criterion mentioned above. It then initializes an internal LIST with the abstract values of the first level of abstraction and those observable values that do not have abstract values associated. If the selected attribute is structured, it selects only one abstract value according to a certain criteria explained below. If the selected attribute is not structured the algorithm selects one of the observable values. Then, EG2 generates a subtree according to this value (abstract or observable). If this subtree is consistent and complete the algorithm stores it in a temporary place because the algorithm will try to get a better generalization than the one stored. But if this attempt does not work, at least it will have the stored subtree.

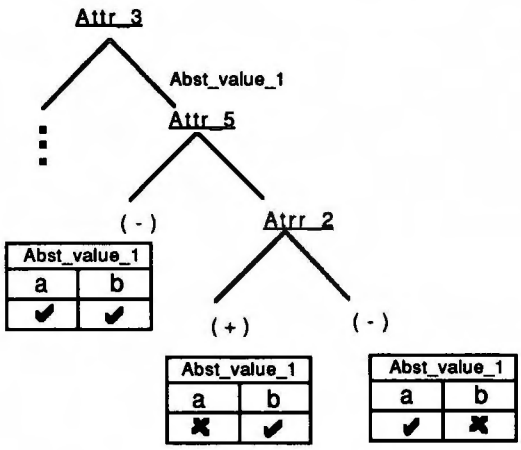
In order to get a better generalization, the algorithm tries the union of the former valid value and other values (abstract or observable), and builds a subtree according to this union. This process is repeated until an inconsistency or incompleteness is detected. If that happens, EG2 recalls the last stored subtree. If an observable value can not be generalized, EG2 generates a subtree according to this observable value (like ID3 does). The process for each selected attribute ends when there is no more values in the internal LIST.

Inconsistency is detected when a leaf cannot terminate, because there are at least two examples in the subset of examples that are described equally but have different classes.

Incompleteness of a generalization is detected by measuring the proportion of its observable values in all leaves of the subtree below the said generalization. This measure should be greater than the threshold predefined by the user. If each leaf contains each of the values of the generalization, the completeness is 1. In general, this factor is calculated as follows:

$$\text{Completeness factor} = \frac{\Sigma i \text{ leaves with i-th value of the Generalization}}{\#\text{ leaves} * \#\text{ values of the Generalization}}$$

The following figure illustrates the detection of the incompleteness of an attribute (attribute_3) in one of its generalizations (a, b are observable values of ABST_VALUE_1). Let us suppose that the user specified a completeness threshold of 0.7.




$$\text{Abst_value_1 Completeness measure} = \frac{2 + 1 + 1}{2 * 3} = 0.66 < ct \quad \textit{(generalization rejected)}$$

Fig. 4. Illustration of an incompleteness detection

The figure 4 shows that the attempt to generalize the values a and b using the ISA hierarchy (Abst_value_1) fails because the completeness measure is lower than the completeness threshold (ct). Then the subtree below the branch Abst_value_1 is destroyed and EG2 will try another abstract value looking at the ISA hierarchy associated to the attribute ATTR_3.

### *3.2 Search of generalizations*

As it was explained above, EG2 tries to find the more suitable UNION of abstract and observable values. In particular, EG2 tries the UNION between the generalization(s) that has(ve) been previously validated and other values not selected before. If a value is selected to share the union, it goes to the head of the internal LIST.

The criteria to select a value to (abstract or observable) in order to put it at the head of the internal LIST are the following, in priority order:

1. Abstract values that belong to a common ancestor in the ISA hierarchy. This process has been called "climbing the generalization tree" (Michalski, 1983)

2. Abstract values with more observable attributes (they are more general).

3. If there is a conflict (there are more than one selected abstract values) OR if there are only observable values then, partition the set of examples into subsets, each one according to each possible generalization. Measure the entropy - $\Sigma$ p.log(p) of the class of each subset of examples and select the generalization that produced less entropy. The purpose of this step is to choose the generalization that best classifies the examples.

The following figure shows the EG2 algorithm:

---

```
EG2 (set_of_examples)

IF all the examples have the same class, then
      Label the leaf with this class
ELSE
      Select attribute.    /* in accordance with the economic criterion ICF*/
      Initialize LIST with the abstract values of the first level of abstraction
            and the observable values that do not have abstract values associated.
      Let Last_valid_generalization = nil; Let Last_valid_subtree = nil
      WHILE LIST is not empty:
            Reorganize LIST
            Construct SUBSET of examples in accordance with:
              Last_valid_generalization   OR  first element of LIST
            EG2 (subset)      /* generate a subtree */
            IF the said subtree is consistent and complete
                  Let Last_valid_generalization = Last_valid_generalization
                                    OR first element of LIST
                  Let Last_valid_subtree =  generated subtree
                  Destroy generated subtree
                  Delete the first element of LIST
                  IF  there is only one element in LIST and subtree is nil
                        Add Last_valid_subtree to the final decision tree
                        Let  Last_valid_generalization = nil
                        Let  Last_valid_subtree = nil
            ELSE
                  Destroy generated subtree
                  IF Last_valid_generalization = nil  AND
                    the first element of LIST is an abstract value:
                        Decompose the said abstract value into its values
                        Let Last_valid_generalization = nil
                        Let Last_valid_subtree  = nil
```

---

Fig. 5 The EG2 algorithm

The following figure illustrates the functioning of EG2:

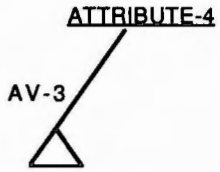
Let us suppose that the selected attribute is ATTRIBUTE_4 and the domain knowledge contains the Abstract Values (AV-1, AV-2 and AV-3) associated to the observable values (a,b,c,d,e and f) of the ATTRIBUTE_4 (note that the value c belongs to two ISA relations):

c,f  ISA  AV-1
a,b  ISA  AV-2
e,d,c ISA  AV-3

The internal LIST is initialized with the abstract values. The algorithm begins executing the subtree corresponding to the first abstract value of the internal LIST. The ordering criteria is explained in section 3.2.

LIST: AV-3 AV-2 AV-1
Execution of AV-3 subtree: consistent/complete.
Therefore, AV-3 disappears form LIST and its observable values (e d c) make disappear other abstract values. Since AV-1 also considers c, AV-1 disappears from LIST and therefore it is decomposed into its values (without c).

LIST: AV-2 f
Execution of AV-3 OR AV-2 subtree: inconsistent
FIRST BRANCH: AV-3

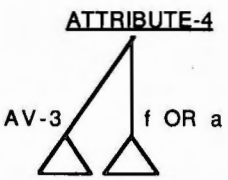



LIST: AV-2 f
Execution of AV-2 subtree: incomplete
substitute AV-2 for values a b

LIST: f a b
Execution of f subtree: consistent/complete
Let Last_valid_generalization: f

LIST: a b
Execution of f OR a subtree:
consistent/complete
Let Last_valid_generalization: f OR a

LIST: b
Execution of f OR a OR b subtree: inconsistent
SECOND BRANCH: f OR a




LIST: b
Execution of b subtree: consistent
THIRD BRANCH: b

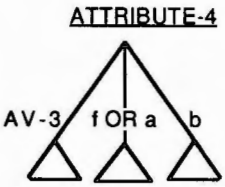



Fig. 6. Illustration of a generalization process executed by EG2

## 4 Results

The following examples have been executed by ALEXIS II program. In the first one, the algorithm applies different types of generalization (climbing the generalization tree and union of values) and optimizes the classification cost. The other examples show the performance of the economic criterion used by EG2 compared with other approaches in two domains.

### *4.1. An example*

In this example (see figure 7 and table 1), the user has supplied the historical data of the problem, and his knowledge about the ISA hierarchy and the cost associated to the attributes. He also has supplied the generalization requirements, that is to say, the completeness threshold (ct = 0.6), and the economic factor ($\omega = 1$).

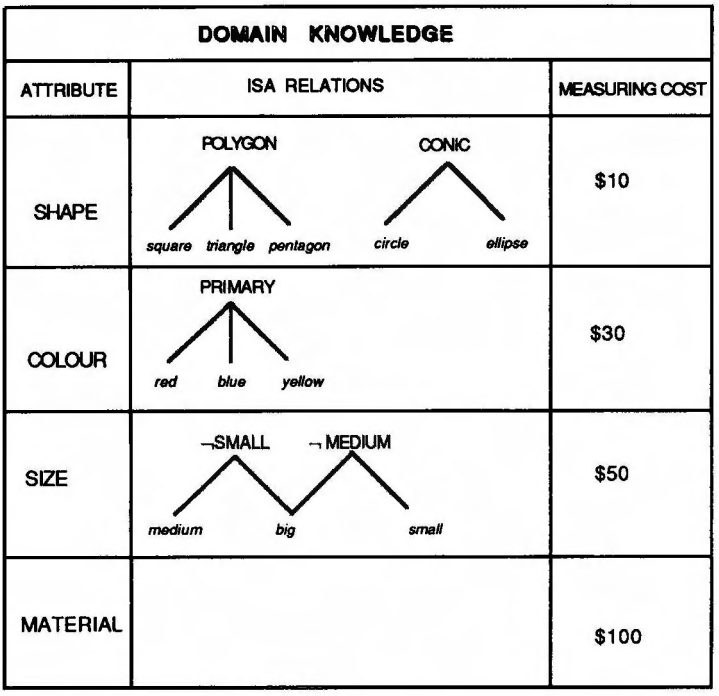



Fig. 7. Domain knowledge of a classification problem

| FORMA | COLOUR | SIZE | MATERIAL | CLASS |
|---|---|---|---|---|
| square | red | big | metal | + |
| square | blue | small | plastic | + |
| triangle | yellow | medium | metal | + |
| triangle | pink | big | leather | - |
| square | pink | medium | leather | - |
| circle | red | small | plastic | - |
| circle | blue | small | metal | - |
| ellipse | yellow | small | plastic | - |
| ellipse | blue | big | leather | + |
| ellipse | pink | medium | wood | + |
| circle | blue | big | wood | + |
| triangle | blue | medium | plastic | + |

Table 1. Training examples of a classification problem

Note that the user has defined the NOT logic operator, indicated with the character "¬" (i.e.¬SMALL, ¬MEDIUM). On the other hand, note that the value pentagon has been defined in

the domain knowledge, however this observable value is not present in the training set. This situation is usual when the user has divided the historical data into two sets: the training and the testing set. ID3 cannot handle this situation. ID3 would classify an instance with shape pentagon as "UNKNOWN CLASS". EG2 can handle this situation because it will try to generalize square, triangle and pentagon as polygons in order to build the decision tree.

The fact that instances with shape pentagon are not present in the training set will decrease the completeness factor of the generalization POLYGON and therefore it is possible that this generalization would fail. Let us see what happens.

In order to select the first attribute, the algorithm measures $\Delta I$ of each attribute, then it applies the economic criterion (ICF) mentioned above.

$$\Delta I = H(TI) - H(NI)$$

$$= \{ -(\frac{7}{12}\log\frac{7}{12} + \frac{5}{12}\log\frac{5}{12}) \} - \{ \frac{3}{12}(\frac{2}{3}\log\frac{2}{3} + \frac{1}{3}\log\frac{1}{3}) * 4 \}$$

$$= 0.061$$

$$ICF_1 = \frac{10 + 1}{2^{0.061} - 1} = 252.27$$

Similar analysis gives

$$\Delta I_2 = 0.119 = ICF_2 = 360.54$$
$$\Delta I_3 = 0.167 = ICF_3 = 415.57$$
$$\Delta I_4 = 0.186 = ICF_4 = 733.98$$

Although the attribute MATERIAL has the highest $\Delta I$, SHAPE has the best relation "cost/benefit". EG2 selects attribute 1 (SHAPE) to build the decision tree, and tries to apply the associated abstract value (POLYGON). Then it selects the attribute COLOUR, and tries its only abstract value (PRIMARY). The following figure shows the decision tree induced by EG2 compared with the decision tree generated by ID3 (or EG2 [$\omega$=0, no domain knowledge]).

Note that the tree generated by EG2 is more economic than than generated by ID3 (i.e. \$ 50 instead of \$ 108), and it is also more general (i.e. it considers pentagons, no present in the training set) . The figure 10 illustrates the economical performance versus $\omega$ and ct.

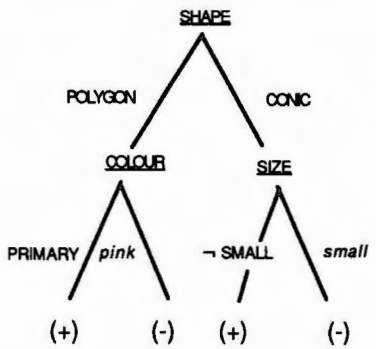



Fig. 8. Decision tree generated by EG2 [$\omega$=1, ct=0.6]
Average classification cost:\$ 50

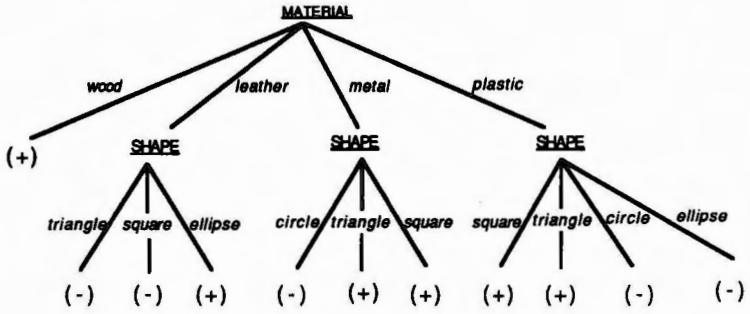



Fig. 9. Decision tree generated by ID3 or EG2 [ω=0, no domain knowledge]
Average classification cost: $108

In general terms, the trees generated by EG2 are more general than those generated by ID3. The "rules" induced by EG2 are confirmed by more examples than the "rules" induced by ID3, therefore the number of leaves of EG2 tree is less than the number of leaves of the ID3 tree (i.e. in the previous example: 4 leaves instead of 11 ). This example shows dramatically that ID3 can not "see" that the polygons have the same classifications or that the conic items also have the same classifications in some circumstances. ID3 can not apply the operator NOT (i.e.¬ SMALL) neither the OR operator.

There is another interesting decision tree that can be generated by EG2: [ω=1, no domain knowledge] (see left side of figure 10 as ct = "1+"). This decision tree has a classification cost slightly lower ($45) than that of figure 8 considering domain knowledge ($50). This is because isolated observable values discriminate better than the generalization of these same values. Thus, in the figure 10, the general decision tree around [ct=0, ω=0] (considering domain knowledge) are more costly than ID3 decision trees, which are more specific. However, a decision tree with generalizations can be more "logical" or easily understood, that is to say, it can best describe a concept. On the other hand, general descriptions can be more precise for testing instances not present in the training set than those less general description.

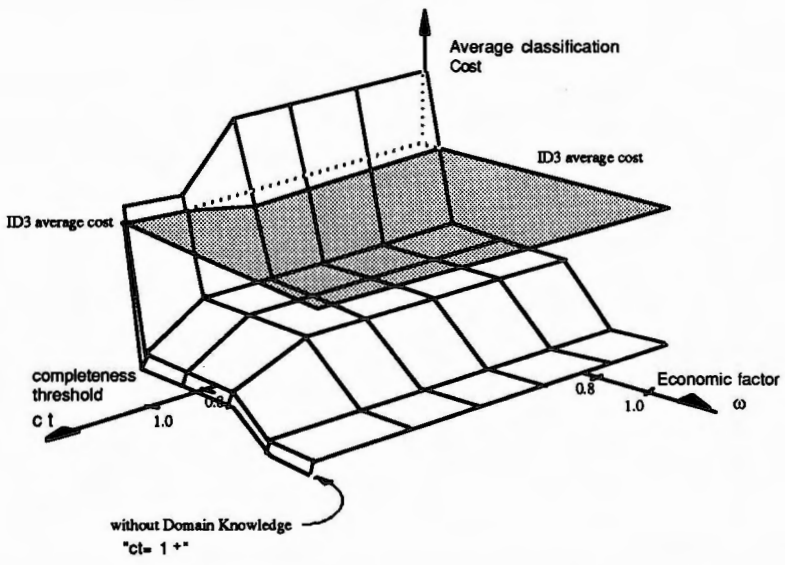



Fig 10. Evolution of the economical performance for any combination of ω and the ct.

### *4.2 Performance of the economic criterion*

In order to test the attribute selection criterion ICF of EG2, some domains were analyzed without considering the domain knowledge. We begin by showing in detail the results obtained in one of these domains (medicine). Then we shall show a comparative table about the economical performance of three attribute selection criteria for building decision trees in two domains.

#### *4.2.1 Gynaecological diagnosis problem*

The learning set of t he Gynaecological diagnosis problem was constituted by 352 cases of women with "amenorrhea" (failure in menses). Each case is described by six attributes and a class. Five classes have been considered, each one indicates an amenorrhea cause: pregnancy, menopause, psychological cause, low grade hormonal failure and "other pathologies". If the program can not give a reliable diagnosis, it classifies the case in "other pathologies", because this application was thought to discriminate the routine, which are the majority of cases, from the complex ones.

Associated with each attribute is an associated measurement cost. The most costly and also effective attribute is the Hormonal Profile Test which classifies every case. Another attribute with a lower cost is the pregnancy test, which can only discriminate the class "pregnancy". The other attributes had no cost. The selection of the first attribute is shown in the following table:

| ATTRIBUTE | $\Delta I$ | $2^{\Delta I}-1$ | COST | ICF |
|---|---|---|---|---|
| Menstruation delay | 0.108 | 0.078 | 0 | 12.82 |
| Age | 0.463 | 0.379 | 0 | 2.63 |
| Early Pregnancy Disorders | 0.081 | 0.058 | 0 | 17.24 |
| β.HCG Pregnancy Test | 0.930 | 0.906 | 2000 | 2207.39 |
| Hormonal Profile Test | 3.142 | 7.83 | 8000 | 1171.44 |
| Coitus Last Period (Yes/no) | 0.365 | 0.288 | 0 | 3.47 |

Table 2. Selection of the first attribute in then gynaecological diagnosis problem.

The information efficiency ($2^{\Delta I}$ - 1) of the Hormonal Profile Test is very high, but it is very expensive, therefore the ICF is high. On the other hand, being the third in information efficiency, "age" has cost = 0.0, therefore the ICF function is very low, and therefore "age" is selected as the first attribute.

The learning set was tested on ID3 and EG2 [ω=1]. The generated trees were very different. ID3 generated a very small tree with just one node in which the selected attribute was Hormonal Profile Test, which discriminated every case. EG2 ([ω=1] generated a larger tree whose first selected attribute is "age" (with symbolic values). It is not a coincidence that one of the datum every gynaecologist needs to know for this particular problem is "age" of the patient, even though he may not ask the patient explicitly.

In order to compare the economic performance of the EG2 economic criterion, the variable ω was set to 1 and to 0 (ID3). The average cost of each diagnosis using ID3 (or EG2 [ω=0]) was 8000 units while it was 3840 units using EG2 [ω=1], that is to say a 52% of economical improvement. In order to make a comparative study, the above cost were modified and its results were analysed in the next section.

#### *4.2.2 Comparative results*

Tan and Schlimmer (1989) proposed an economic criterion to construct decision trees. According to this economic criterion the algorithm selects the attribute with the highest $\Delta I^2$ / cost. This criterion also constructs decision trees with less classification cost than ID3.

In order to compare the three attribute selection criteria (ID3, EG2, Tan and Schlimmer's), two domains were analyzed. The Gynaecology domain (problem a) analyzed above and another one about classifying flags (Forsyth and Rada, 1986) for 10 different cost situations. The following table shows the economic performance of the three attribute selection criteria in two domains, for 10 different assignments of measuring attribute costs:

| PROBLEM | ID3 | Tan & Schlimmer´s | EG2 [ω=1] |
|---|---|---|---|
| Cost situation a.1 | 8000 | 4532 | 3840 |
| Cost situation a.2 | 4362 | 3576 | 2312 |
| Cost situation a.3 | 1501 | 1201 | 930 |
| Cost situation a.4 | 4157 | 3907 | 3907 |
| Cost situation a.5 | 119 | 98 | 86 |
| Cost situation a.6 | 73 | 55 | 46 |
| Cost situation b.1 | 211 | 141 | 84 |
| Cost situation b.3 | 14026 | 12489 | 8365 |
| Cost situation b.3 | 27092 | 18029 | 14903 |
| Cost situation b.4 | 35484 | 30163 | 27679 |

Table 3. Comparative table that shows the average classification cost of three criteria for selecting attributes.

The percentage shows how much cheaper each decision tree is by using an evaluation function (Tan and Schlimmer approach and EG2) compared with the tree generated using ID3. The table shows that the decision trees generated by EG2 are cheaper than or equal to the decision trees generated by Tan and Schlimmer approach.

Tan and Schlimmer approach is closer to ID3 than EG2 [ω=1], approach. The average classification cost of the Tan and Schlimmer approach is between EG2 [ω=1], and ID3, and the average depth of the generated tree is also between EG2 [ω=1], and ID3.

## 5. Conclusion

Domain knowledge is a necessary component of inductive learning, as it has been proved in previous systems (Michalski, 1983; Kodratoff, 1986). The method described uses domain knowledge to provide information to perform economic induction, and to provide constraints and guidance to build a more general and suitable decision tree.

The proposed method also focuses on economy of resources. Economy is a very important part of common sense reasoning and should be included in learning systems.

Among specific measures for evaluating the "quality" of inductive assertions, Michalski (1983) mentioned the cost of measuring values, however he added that mixing cost analysis and generalization are computationally costly for his Star methodology. The proposed approach to perform economic induction and to generalize from domain knowledge is effective, fast, and easy to program and use.

## 6. Acknowledgements

I am grateful to Jaime Carbonell for his useful comments and suggestions. I would like to thank Luis Esteban García from ETSIT (Universidad Politecnica de Madrid) who programmed ALEXIS II and help me analizing the results, to Zandra Montañez and Luz A. Uribe M.D. from Clínica de la Concepción for formalizing and providing data for the Gynaecology domain. I am also grateful to Emilio Rodriguez for his comments.

## 7. References

Cendrowska, J. (1989). PRISM: An algorithm for inducing modular rules.Knowledge Acquisition for Knowledge Based Systems (Vol . 1). Gaines B. & Boose J. eds. Academic Press.

Breiman, L. Friedman, J., Olshen, R & Stone, C.J. 1984. Classification and Regression trees. Belmont: Wadsworth.

Duda R. et al. (1981). Model Design in the Prospector Consultant System for Mineral Exploration, Expert Systems in the Microelectronic Age, D. Michie (ed), Edimburgh Scottland, Edimburgh Univ. Press.

Gaines, B., 1989. Knowledge Acquisition: The Continuum Linking Machine Learning and Expertise Transfer. Proceedings of the EKAW 89. Boose J., Gaines, B., Ganascia, J. (eds) .

Forsyth R. and Rada R. (1986). Machine Learning Applications in Expert Systems and Information Retrieval (pp 60-64), Ellis Horwood Publishers, West Sussex, England.

Hunt E.B., Marin J. Stone P.J. (1966). Experiments in induction, New York, Academic Press.

Kodratoff Y. and Ganascia J. (1986). Improving the Generalization Step in Learning, Machine Learning Vol 2, Michalski R.S. Carbonell J.G. Mitchell T.M. eds., Morgan Kauffman, Los Altos

Kononenko, I., Bratko, I., & Roskar, E. 1984. Experiments in automatic learning of medical diagnostic rules. Technical Report, Jozef Stefan Institute, Ljubljana, Yugoslavia.

Lavrac, N, Mozetic, I & Kononenko, I, 1986. An Experimental comparison of two learning programs in three medical domains. Proceedings of ISSEK Workshop 86. Turing Institute, Glasgow.

Larson and Michalski R. (1977). Inductive Inference of VL Decision Rules, Proceedings of the WPDIS-77, Sigart, Newsletter 63. Michalski R. (1983). Theory and Methodology of Inductive Learning,, Machine Learning Vol 1, Michalski R.S. Carbonell J.G. Mitchell T.M. eds., Morgan Kauffman, Los Altos (ca).

Nuñez, M. (1988a). El Metodo de Aprendizaje EG2: Una Aplicacion de Conociminento de Base a Ejemplos Estructurados, Master Thesis in Knowledge Engineering, Universidad Politecnica de Madrid.

Nuñez, M. (1988b). Economic Induction: A Case Study, in Proceedings of the Third European Working Session on Learning, Derek Sleeman (ed), Pitman Publishing, London.

Quinlan. J.R. (1979). Discovering Rules by Induction from Large collection of Examples, in Expert Systems in the microelectronic Age, D. Michie (ed), Edimburg University Press.

Quinlan. J.R. (1986). Induction of decision trees. Machine Learning, 1,1.

Quinlan. J.R. (1989). Simplifying decision trees. Knowledge Acquisition for Knowledge Based Systems (Vol . 1) . Gaines B. & Boose J. eds. Academic Press.

Shannon C.E & Warren Weaver. (1971). The Mathematical Theory of Communication, University of Illinois Press.

Tan M. and Schlimmer J. (1989). Cost-Sensitive Concept Learning of Sensor Use in Approach and Recognition, in the Proceedings of the Sixth International Workshop on Machine Learning, Ithaca, New York.

Winston, P. H. (1977). Artificial Intelligence, Addison-Wesley.

# Towards Knowledge Acquisition From Domain Books*

Lu Ruqian Cao Cungen
Institute of Mathematics
Academia Sinica
Beijing, P. R. China

Abstract. This paper introduces an approach of knowledge acquisition—— acquiring knowledge from domain books. Also presented is a "Rapid Prototyping and Refining" technique for developing expert systems. The adaptability of ES development tools is discussed in some detail.

## 1. Introduction

With the advancement of expert systems and other knowledge—based intelligent softwares, a critical bottleneck has occurred in the knowledge acquisition phase. It is widely accepted that the effective development of a knowledge—based system depends heavily on the efficiency of its knowledge acquisition.

It is very hard to dig out the knowledge directly from an expert. so, we need some tools.

During the past two decades, various methodologies and tools of machine learning (ML) have been studied and implemented, and some ML tools have been reported quite successful [1, 2].

A ML tool is promising in the sense that it can learn (usually induce) the expertise and experience of an expert from a set of well—formed examples provided by the expert. A ML program works based on a package of knowledge about the discourse. Where does such a package come from? Usually, it is collected by knowledge engineers from textbooks and experienced professionals.

Nevertheless, a conventional ML system is rather superextant in the sense that what it can learn is just a set of rules which, however, can not really reflect the expert's know—how; and also it is difficult for it to learn many sorts of knowledge necessary for a practical knowledge—based system, such as causalities, associations between attributes, and definitions and explanations of terms.

It is widely admitted that, in order to be properly used by a user, an expert system should be instructive and interactive; in other words, it ought to explain to the user what it is doing (has done), how it has done and justify why it is doing (has done) in a certain way, whenever the user expects to know the reasoning behaviour of a running expert system.

To answer the user's inqueries like those requires a kind of knowledge, called deep knowledge. Deep knowledge of a domain can be used to analyze and explain phenomena and findings of that domain. Deep knowledge is not easy to be acquired directly from domain experts, but in domain books, we can easily take it.

In recent years, we have been concentrating on the research of knowledge acquisition from books and textbooks [3, 4, 6].

There are at least two approaches for eliciting knowledge from domain books [4].

---

* The research is partially supported by the TWAS grant RG 86—79 and a special grant of Academia Sinica.

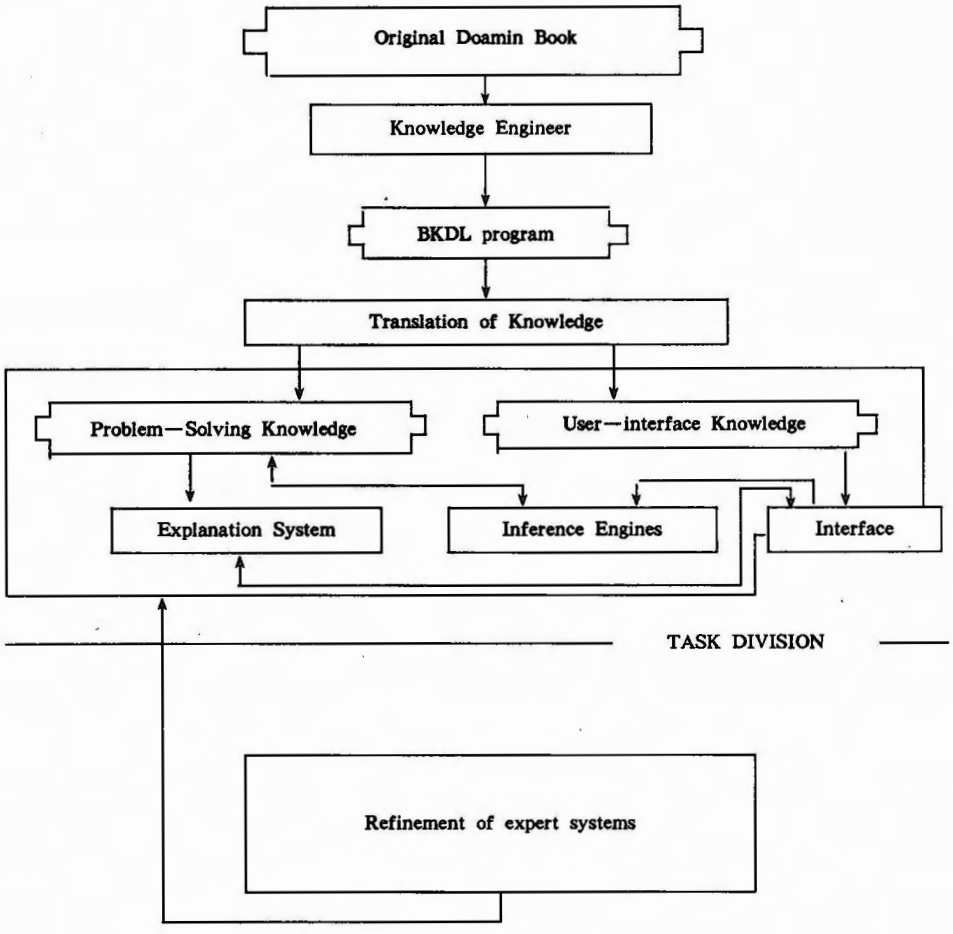



Fig. 1 : Rapid Prototyping and Refining of Expert Systems

One is natural language understanding; but as we know, this path is full of setbacks and obstacles. We have taken the other pragmatic way, i. e. two—step knowledge translation: Firstly, a knowledge engineer rewrites a book in a certain book knowledge description language(BKDL), and secondly the BKDL "book" is translated and re—arranged into a knowledge base by a program.

After this knowledge base(KB) has been assembled with a pre—implemented inference engine, an expert system is produced. Sometimes, the produced expert system is coarse in the sense that it lacks experts' experience. Hence, it needs refining(figure 1).

In the approach of ES development shown in figure 1, the whole of knowledge acquisition is divided into two relatively dependent phases, which are indicated by the line. The work below the line(i. e. the second phase) is to refine the knowledge obtaineded during the first phase. The refining task can be finished by a domain expert, or a knowledge engineer through domain examples.

In 1987, we implemented a tool of refining expert systems, called KBRS[7]. In early 1989, we implemented another tool, called CONBES, which was used to build "book" expert systems.

The core of CONBES is BKAS2. BKAS2 includes two main parts: BKDL, a user language for describing books, and BKTS, a system for translating BKDL programs into knowledge sub—bases. BKDL incorporates a built—in natural language frame, which makes its pro-

grams look very much like Chinese texts while keeping the semantic unambiguity. BKTS is capable of differentiating surface and deep knowledge within the BKDL program, and organizing them separately into layered knowledge sub—bases.

The method of taking knowledge from books is distinguished in five dimensions:

(1). Sharply relaxing the burden of knowledge engineers in acquiring domain expertise. In the development of an expert system, knowledge engineers usually have to read domain books and collect knowledge from them. In CONBES, these tasks can be done by BKTS.

(2). Loosening the dependency between domain experts and knowledge engineers. The efficient development of an expert system requires effective cooperation between kwonledge engineers and domain experts, but the cooperation usually adopted is rather ill—defined[1].

(3). Taking from books many sorts of knowledge, which are necessary and important for an practical expert system and are not so easy to be learned by a conventional ML system. Among them are meta information, causalities, associations between attributes. It is also worth mentioning that the majority of the knowledge owned by a practical expert system can be elicited from domain books[3].

(4). Shortening the life cycle of an expert system development.

(5). Eliciting knowledge from domain books supports the domain adaptability of CONBES. When an expert system is produced with CONBES, it has a domain—specific user interface, rich expertise, and three reasoning modes.

Since the completion of BKAS2, we have been making significant improvements on BKAS2, BKDL and BKTS. The new versions of BKAS2, BKDL and BKTS will be called BKAS2.1, BKDL1.1 and BKTS1.2, respectively.

This paper consists of three main parts, i.e. BKDL1.1 language, BKTS1.2's knowledge base, and BKTS1.2, a translator from BKDL1.1 to its knowledge base.

## 2. BKDL1.1 language

BKDL1.1 is a subset of Chinese, and it is a descriptive language. The structure of BKDL1.1 is a nested one, with each block called a BKDL1.1 chapter(or chapter for short). A chapter is composed of BKDL1.1 sentences and BKDL1.1 subject blocks, most of which are chapter—free.

### *2.1. BKDL1.1 chapters*

The books are a common carrier of human knowledge. While being written by an author, the contents are always deliberately arranged and located in order to be easily read and understood. So, careful readers may find that the presentations of a book usually appear in compact chunks.

Another factor which contributes to that characteristic is the latent hierachy of knowledge. Putting the above analyses together, we have introduced in BKDL1.1 the concept "chapter", to which the naturalness of BKDL1.1 owes much.

A chapter serves to describe a closely compact part of an book. BKDL1.1 includes fourteen chapters, and these chapters can express nine types of compact parts of a book, such as causalities, knowledge structures, examples,mechanisms of physical objects, descriptions of goals and attributes.

Among the fourteen chapters there is a special chapter, called "discrete chapter". A discrete chapter is used to describe knowledge chunks which are not suitable in form to be expressed by other chapters. The discrete chapter is the "most powerful chapter" in the sense that it can represent whatever other chapters can. But it does not possess its own stable and fixed grammatical structure.

Each chapter, except the discrete one, is composed of two parts. The first part is a introductionory sentence(a discrete chapter has no introductionory sentence). The second one is a

chapter body, which is composed of BKDL1. 1 sentences and subject blocks. Note that a BKDL1. 1 chapter doesnot necessarily correspond to a chapter of a domain book; in other words, it may correspond to a chapter, a section or a paragraph of the original book. The Appendix will present part of the English version of the program translated from [5], and the second chapter of the presented program body is a BKDL1. 1 classification chapter which, plus the omitted description, describes a section of [5].

### 2. 2. *BKDL1. 1 subject blocks*

BKDL1. 1 provides eleven subject blocks. A subject block is smaller in structure than a BKDL1. 1 chapter. It may be considered as a special compound sentence, and it serves to describe those knowledge units which belong to the same facet(i. e. subject) of a goal or hypothesis. It consists of three parts——the subject name, subject—body link and subject body.

BKDL1. 1 subjects include 事实[fact]*,性质[property], 观察[observation],特点[characteristic],发现[finding], 检查[examination], etc. Among the subject—body link are 有[there to be], 包括[to include], 是[to be], 为[to be], 表明[to show],体现在[to be embodied in]. The function of a link is of syntax rather than of semantics. A subject body may be composed of simple sentences, compound sentences and its son subject blocks. An example of the subject blocks is given as follows:

> BKDL1. 1 的 特点 包括: 高度类自然, 灵活, 模块化, 领域适应性, 可扩充, 容歧义. [The characteristics of BKDL1. 1 include: High naturalness, flexibility, modularity, domain adaptability, extendibility, and ambiguity tolerance. ]

Moreover, users can define new subject blocks if they have defined new subjects.

### 2. 3. *BKDL1. 1 sentences*

The BKDL1. 1 sentences fall into five categories:

— Elementary sentences: Elementary sentences are used to describe simple facts and declarations. The sentences of the elementary category are broken down into nine types, such as EXIST—TYPE, PASSIVE—TYPE, BE—TYPE, SUBJECT—PREDICATE—TYPE, STRING—TYPE. An elementary sentence may or may not contain keywords. For example, in the BE—TYPE sentence 表面 是 平滑[The surface is smooth], 是[is] is a system—level keyword. But the sentence 机器 运转 正常[The machine is running normally] contains no keywords.

— Compound sentences: Compound sentences are composed of simple sentences through BKDL1. 1 keywords. They can represent compound facts and knowledge chunks.

— Fixed sentences: Fixed sentences serve to express the Chinese sentences whose syntactic structures are relatively fixed and stable.

— Commentary sentence: It functions as the commentary sentence of any algorithmic language. In a commentary sentence, comments are delimited in braces.

—Introductory sentences: They are used to introduce BKDL1. 1 chapters. For example, the introductory sentence 整数 分类为 负整数, 零, 正整数[The integers are classified into the negative integer, zero, positive integer] serves to introduce a classification chapter about integer description;

### 2. 4. *BKDL1. 1 keywords*

BKDL1. 1 keywords are basic semantic elements of BKDL1. 1. BKDL1. 1 has about 250 keywords. They are divided into two levels, i. e. the system and domain levels (The division is more according to their functions in expert systems than according to their meanings in Chinese). A user can utilize them to define his own keywords in the head or body of a BKDL1. 1 program.

---

* Henceforth, if necessary, a BKDL1. 1 keyword or sentence will be followed by a bracketed English equivalent.

A few members at the system level carry special meanings. In Chinese, for example, the words 如果, 倘若 and 若 have the same meaning(their English equivalent is "IF"). But in BKDL1. 1, 若 means that the information unit following it can be obtained only at a high cost(e. g. a major medical operation);如果 at a low cost; what follows 倘若 is a heuristic rule.

*2. 5. BKDL1. 1 programs*

A BKDL1. 1 program is composed of two grammatical parts, namely the program head and program body. The head includes four types of declarations, which are defaultable:

(1). Program name.

(2). Background of anexpert system: This part includes a dozen of items about the expert system to be built, such as 系统名称[title of expert system],成型人员[designer], 成型日期[date], 原书名称[name of original book], etc.

(3). Domain—specific specifications and definitions: Users can specify user—level keywords in this type(they can also specify them in the program body).

(4). Specifications of term unification: In natural languages, words may have several synonyms. Users can indicate synonyms in the program head(they can of course indicate them in the program body). If a user is unwilling to specify them in his BKDL1. 1 program, he can take another(but final) chance: During the interaction procedure of BKTS1. 2, he can interact with BKTS1. 2 to unify the terms (i. e. attributes and their values).

A BKDL1. 1 program is composed of BKDL1. 1 chapters(see Appendix).

*2. 6. Characteristics of BKDL1. 1*

BKDL1. 1 possesses many noticeable qualities and fectures, and some of them are presented in the following:

— Representional adequacy BKDL1. 1 aims mainly at describing knowledge about classification problems. It can naturally represent the following types of knowledge:

- Meta information;
- Causalities;
- Planning Knowledge about classification;
- Functions and mechanisms of physical objects;
- Definitions and properties of physical objects;
- associations between attributes;
- Goal descriptions;
- Classification examples;
- Knowledge tables.

— Flexibility The mapping from BKDL1. 1 to Chinese is one—to—many. When a user uses it to translate a book, he does not need to change the original presentations greatly. He can specify his keywords and knowledge tablesin the head or body of the BKDL1. 1 program. If he doesnot feel a BKDL1. 1 keyword agree with his linguistic usage, he can re—specify it in the program. Moreover, he can easily "write" his experience in BKDL1. 1.

— Modularity BKDL1. 1 is a nested and structured language. The modularity reflects the local compactness of books.

— Ambiguity Tolerance Each BKDL1. 1 keyword carries its definite semantic information which is quite close to its original one in Chinese. However, ambiguities are allowed in a BKDL1. 1 program, which may be resolved according to their contexts by BKTS1. 2. For instance, 也称[to be also named] and 又叫[to be also called] are two BKDL1. 1 keywords. 也称 connects a synonym with a goal, and 又叫 joins a synonym with a term. But, 也称 and 又叫 may be allowed to ambiguously represent a synonym relation either between two goals or between two terms, leaving the ambiguity to be resovled by BKTS1. 2.

—Extendiblity In BKDL1. 1, the keywords are divided into three levels, namely the system level, domain level and user level. A user can use the keywords at the system level to

specify and define his own keywords of the domain level, keywords at the domain level to define keywords of the user level. Furthermore, the fixed BKDL1. 1 sentences containing two BKDL1. 1 keywords hold a kind of capability in defining new keywords or indicating more domain—specific ones which have already existed in the keyword lexicon of BKDL1. 1.

During its translation phase, BKTS1. 2 can save user—defined keywords and dig out more domain—specific ones from the fixed sentences of the BKDL1. 1 program, so that the user can directly apply them in next BKDL1. 1 programmings.

— Domain adaptability Domain adaptablity is an obscure criterion for a tool of expert system development. It may be informally measured by the easiness for a general—purpose tool to become domain—specific. Usually, general—purpose tools and programming environments throw the consideration of domain adaptability over to their users. The characteristics of BKDL1. 1 mentinoned above obviously contribute much to the domain adaptablity of BKDL1. 1, on which the domain adaptability of CONBES is based. When an expert system is produced through CONBES, it possesses not only several forms of knowledge representation, but also a domain—specific user interface and three reasoning modes.

## 3. BKTS1. 2's knowledge base

### *3. 1. The functional structure of knowledge*

The structure of any knowledge can be described from two viewpoints. The first one is the static structure, which will be treated of in Section 3. 2. The second one is the functional structure. A book dealing with classification tasks and problems holds many kinds of knowledge, and each of them, in terms of expert systems technology, has its own functions, which are illustrated in figure 2.

In figure 2, ·····> means the relation "TO EXPLAIN", and→ stands for the relation "TO BE USED TO SOLVE".

### *3. 2. The representational forms of knowledge*

BKDL1. 1 can describe many sorts of knowledge, and correspondingly, BKTS1. 2 provides several representational structures of knowledge. The representational forms of any knowledge should not only support its functonal structure, but should reflect its static structure as well.

BKTS1. 2's knowledge base can hold descriptions about goals, examples, causalities, atrributes, associations between attributes(including fact—fact rules), fact—goalrules, fact—hypothesisrules, terms, user—interface vocabulary, etc. Main representational forms of knowldge are given as follows:

#### *3. 2. 1. Goal frames*

In BKTS1. 2, a goal is a concept in the hierachy of concepts. Goals are represented by frames, whose structure is given in table 1. The slots of a goal

Table 1: Structure of a goal frame

| | |
|---|---|
| GOAL—NAME: | <goal name> |
| FATHER: | <goas> |
| CHILDREN: | <goals> |
| CONCURRENT—WITH: | <goals> |
| EXCLUSION: | <goals> |
| DEVELEPING: | <goals> |

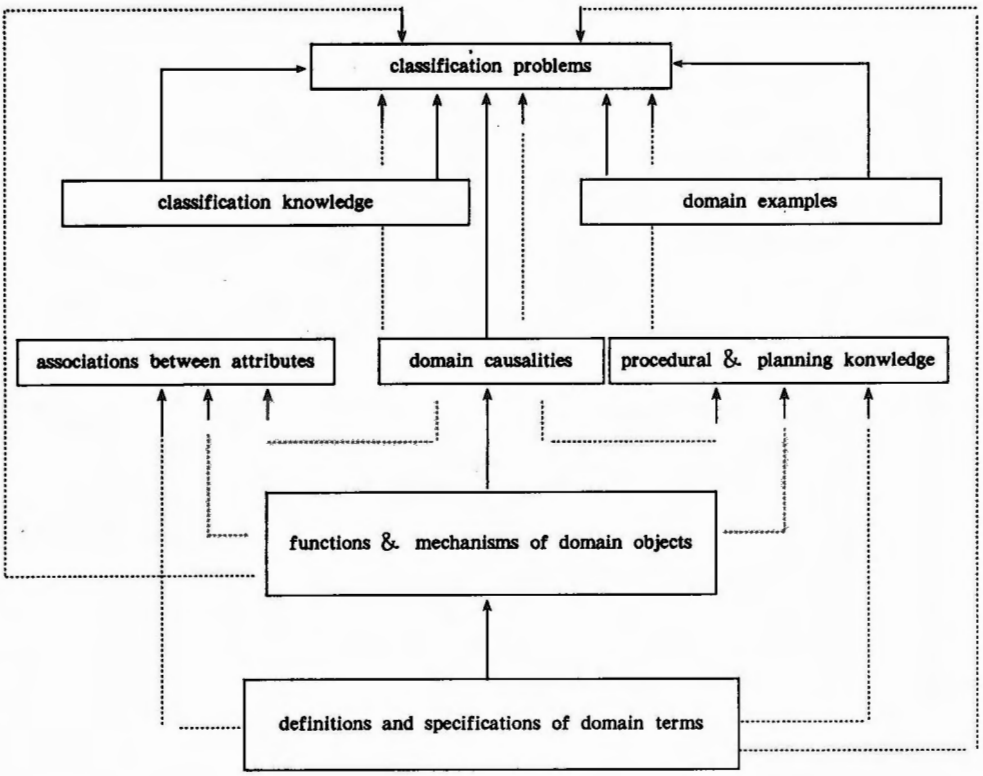



Fig. 2: Functional structure of knowledge

DEVELOPED—BY: <goals>
ASSOCIATED—WITH: <goals>
DIFFERENTIAL: (< premises ——> goals>, where premises are and/or normal formulae of <attributes>)
CLADESCR: <classification descriptions of a goal>

| cluster 1 | ... | cluster n |
|---|---|---|
| CLUSTER—NAME: <cluster name> | | CLUSTER—NAME: <cluster name> |
| CVALUE: 0——1 | | CVALUE: 0——1 |
| NECESSARY—ATTRIBUTES: <attributes> | | NECESSARY—ATTRIBUTES: <attributes> |
| SUFFICIENT—ATTRIBUTES: <attributes> | | SUFFICIENT—ATTRIBUTES: <attributes> |

frame are broken down into three parts:

— Goal name.

— Relations between goals: Several relatons are captured in this part. The slot DEVELOPING(DEVELOPED—BY) is a temporal relation rather than a causal one. Concurrency is a common natural phenomenon, and a goal frame can represent concurrent relations between goals in the slot CONCURRENT—WITH.

— Classification descriptions about a goal: This part includes many clusters of attributes, since a goal may be described in many facets. Usually, the slot CLUSTER—NAME is null. The intersection of cluster i and cluster j is not necessarilyempty. CVALUE carries a threshold which indicates the confirmation of a goal if the matching score of cluster i is equal to or greater than it.

### 3. 2. 2. *Hypothesis frames*

Table 2: Structure of a hypothesis

| | | |
|---|---|---|
| DESCRIPTION: | HYPOTHESIS—NAME: | <name of a hypothesis> |
| | CVALUE: | 0——1 |
| | IMPORTANCE: | <measurement of importance> |
| | MODIFICATIONS: | <modifications of a hypothesis> |
| | EXPLAINING: | <goals> |
| | ATTRIBUTES: | <attributes of a hypothesis> |
| | COST: | <cost at which the entity or occurrence of a hypothesis is directly detected> |
| DIFFERENTIAL: | | <and/or formulae of hypotheses——>a hypothesis> |
| CONCURRENT—WITH: | | <hypotheses> |
| EXCLUSION: | | <hypotheses which are mutually exclusive with a hypothesis> |
| CAUSED—BY: | | <hypotheses> |
| STRENGTH—OF—CAUSE: | | <strength measurement of the cause> |
| CAUSE—OF: | | <hypotheses> |
| STRENGTH—OF—EFFECT: | | <strength measurement of the effect> |

Hypotheses are represented by hypothesis frames whose formula is given in table 2. A hypothesis is different from a goal, and it can be considered as a node in the causal network. The slots of a hypothesis frame fall into two sections:

— Descriptions of a hypothesis: This section is represented by a description frame. The slot EXPLAINING indicates that the hypothesis gives a partial explanation to the goals involved, and the overall explanation of a goal is given jointly by those hypotheses which give partial explanations to the goal. ATTRIBUTES are direct concequences of a hypothesis.

— Relations between hypotheses: Several relations are included in this part. Through these relations, hypotheses are linked with each other to form a semantic network of causal flavour.

### 3. 2. 3. *Attribute frames*

An attribute is a descriptor of a goal. An attribute is usually modified by several properties, and has many associations with other attributes. In an attribute frame, ASSOCIATION—TYPES may carry a dozen of information about reseaning controls as well as associations between attributes.

MODIFICATIONS are further descriptions of an attribute. Among the modifiers are STATUS, HISTORY, CURRENT, FUTURE, DEGREE, MODE, STATE, LOCATION, DIRECTION, DURATION, CONCURRENT — WITH, EXCLUSION, etc. STATUS is a flag, which indicates the presence or absence of an attribute. In addition, users can define modifiers in BKDL1. 1 programs.

### 3. 2. 4. *Dictionary of terms*

This dictionary is used to hold the explanations and specifications of domain terms. Usually, the structure of the dictionary is a forest. Currently, the mechanisms and functions of domain objects are treated as terms by BKTS1. 2, and are recorded in this dictionary.

### 3. 2. 5. *User—interface lexicon*

Before BKTS1. 2 works, the user—interface lexicon contains all the keywords of the

Table 3: Structure of an attribute frame

| | |
|---|---|
| ATTR—NAME: | <attribute name> |
| VALUE—TYPE: | (single, multiple, yes—no) |
| VALUES: | <attribute value(s)> |
| MODIFICATIONS: | <modifiers that further specify an attribute> |
| ASSOCIATION—TYPES: | <list of types of associations> |
| ASSOCIATION—VALUES: | <list of attributes> |
| BELIEF—MEASUREMENT: | < belief measurement of an attribute> |

Table 4: An item of the user—interface lexicon

| | |
|---|---|
| S—KEYWORD: | <a system—level keyword> |
| AGENT: | <an agent> |
| AGENT—ID: | (unchanged, user—defined, domain—level) |

(Note that AGENT—ID is a flag, which indicates whether the AGENT is changed or what change has occurred to it.)

system level, most of which are each attached with one domain—level keyword, called an agent. During the grammatical analysis of BKTS1.2, an agent is either unchanged, or replaced a user—defined keyword or a domain—specific keyword of the domain level which is digged out of fixed sentences in the BKDL1.1 program.

The user—interface lexicon is not a dispensable trick. Instead, it plays an important role in making domain—specific the user interface of an expert system produced by CONBES.

## 4. BKTS1.2: a translator from BKDL1.1 to its knowledge base

The semantics of a BKDL1.1 program is reflected by the semantics of BKDL1.1 and user—defined keywords, and according to them, the evaluation and linking of representational frames and dictionaries are achieved by BKTS1.2. BKTS1.2 is in fact the BKDL1.1 compiler. The compilation includes three phases(figure 3). Unlike the compiler of an algorithmic language, BKTS1.2 is interactive. The routines for collecting and linking knowledge scattered in the BKDL1.1 program are its kernal components.

### *4.1. Pass One: the first phase*

#### *4.1.1. Analysing the program head*

This first step includes parsing(both grammatical and semantic), changing the user—interface lexicon, and filling up the synonym table and other information tables.

#### 4.1.2. *Analysing the program body*

The program body is analysed both grammatically and semanticly; and the slots of representational frames are partially evaluated and re—arranged.

When an attribute frame is generated, the collecting and linking procedure is responsible for connecting it to the goal or hypothesis frame involved and linking it with other attributes of the goal or hypothesis which have been already collected. When a goal(hypothesis) frame is partially instantiated, it is linked to other goal(hypothesis) frames to form a network of goals(hypotheses).

During this analysis, the user—interface lexicon, the synonym table and other representational forms are further filled in.

### *4.2. Interaction: the second phase*

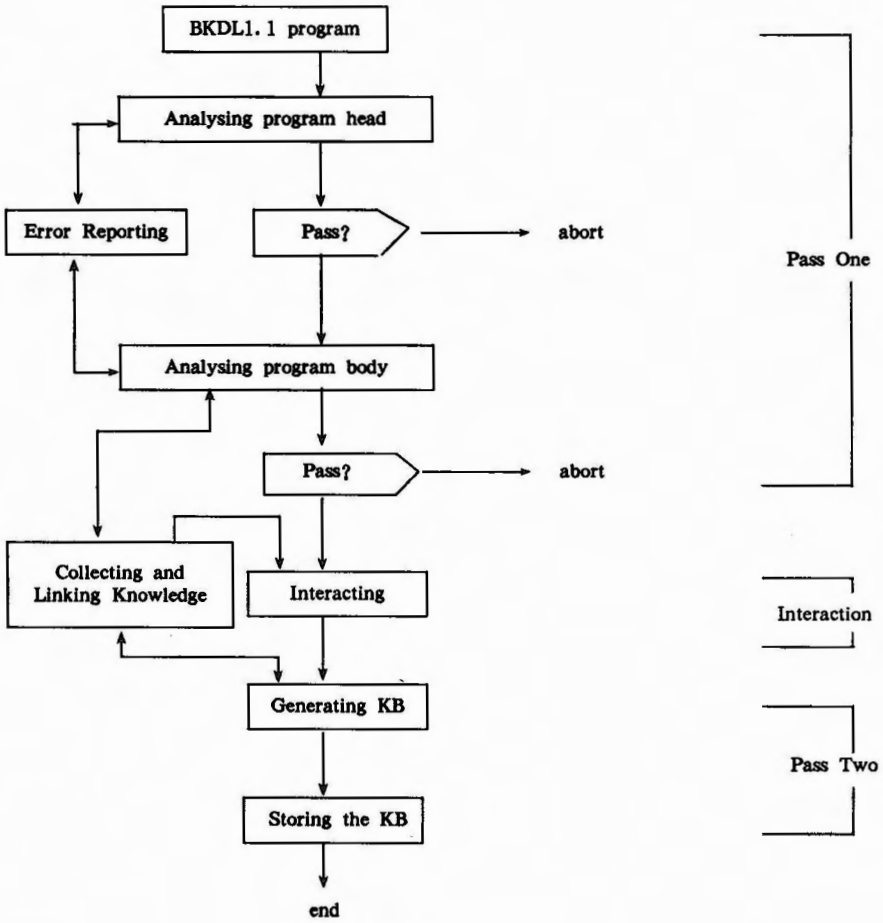



Fig. 3 Flow diagram of BKTS1.2

The second phase is an interactive one; that is, during this phase BKTS1.2 helps the user see whether the structure of goals is a connected network (if not, then BKTS1.2 helps the user connect it), and whether an attribute has two or more values with the same meaning (if so, then BKTS1.2 helps the user rule out the reduntant ones). In addition, BKTS1.2 asks the user whether he wants to add the keywords he defined in his program to the temporary keyword base of BKDL1.1. If he wants to, then BKTS1.2 add them to their brothers or sons, respectively; thus he can utilize them in other programmings.

*4.3. Pass Two: the third phase*

During the third phase the rest of all representational forms are filled in, and the causal chains generated in pass one are further linked. Meanwhile, various relations between knowledge units are analysed and then tied, and conflicts in the knowledge base are recorded, which will be resolved by KBRS (see figure 1).

## 5. CONBES's scheduling of knowledge

three modes of reasoning. The reasoning of the first mode is a bidirectional, and it is done only by the surface inference engine, which works based on the network of goal frames. The reasoning of the second mode is done just by the deep inference engine, and it is based on the semantic network of hypotheses. The third mode provides a mixed surface—deep inference, which is based on both the network of goal frames and the network of hypotheses, and finished jointly by the deep engine and the surface engine. The second and third modes of inference are necessary for a real expert system, because they agree in many ways with reality(e. g. in focusing and shifting of inference).

It is worth mentioning that many disciplines have not developed and advanced well enough, thus suffering a shortage of deep knowledge. That is why we have designed in CONBES two inference engines and three reasoning modes.

During the information acquisition of inference, the ASSOCIATION—TYPES and ASSOCIATION—VALUES slots of attribute frames play an important role in controlling the generations of interaction menus and making the process of information acquisition intelligently ordered.

At each step of reasoning, the user can ask the explanation system to give reasonable justifications or explanations of activities of the inference engine(s)(refer to figure 2)

## 6. Final remarks

The described BKDL1. 1 language and BKTS1. 2 are our work along the line of knowledge translation. Knowledge translaton is a universal concept, and our method of acquiring knowledge from technical books is just a case of that concept. Furthermore, the theory of knowledge translation can construe a general scope of the behaviours of human beings and machines. For example, human learning is a case of knowledge translation, and it is a translation from the external environment to the brain. In AI, various techniques of knowledge acquisition are also examples of knowledge translation. Among its phases, software engineering can see many phenomena about knowledge translation.

Translators of knowledge translation may be human beings, machines, or both. In our case, the translators are: knowledge engineers, who do the first translation(from Chinese to BKDL1. 1), and BKTS, which finishes the second translation(from BKDL1. 1 to BKTS1. 2's knowledge base).

The domain adaptabilty of a general—purpose tool is discussed in this paper. It should be considered seriously when we design ES development tools, since too "universal" tools are not easy to be applied by their users. The users usually expect tools to be domain specific rather than general—purpose.

KBAS2. 1 is coded in the svs PASCAL,and KBRS is written in the svs C language. They are running on a UV68000 microcomputer under the UNOS operating system.

We have constructed two systems with CONBES. The first one is a Chinese medical consultant, called DIAA. Its KB is generated from a BKDL1. 1 program which is a translation of the book written by Wu Boping, a specialist on Chinese medicine. The second one is a matchmaker, called DSTA, which can help bachelors and maidens determine their spouse types according to their personalities and qualities. DSTA is based not on any book, but on our social experience, which means that BKDL1. 1 is also suitable to describe personnal experience.

The current BKDL1. 1 requires elaboration and expansion. Among the issues that are most important and need further attention is: To extend BKDL1. 1's abilities in describing procedural knowledge and general planning knowledge.

Appendix: part of a BKDL1. 1 program

This appendix presents part of the English version of the BKDL1. 1 program translated from [5]. The words underlined may been considered as system—level keywords of BKDL1.

ed from [5]. The words underlined may been considered as system—level keywords of BKDL1.1.

Building ES Based On <<Pathology>>(By Prof. Wu Zhongbi)

Declarations

ES—Background
Name: a Clinical Consultant of Hepatocirrhosis;
Author: Cao Cungen;
Date: March,1989;
Copyright: Institute of Mathematics, Academia Sinica. Specific
Goal: hepathocirrhosis.

{BEGIN OF PROGRAM}

Begin

{BEGIN OF DISCRETE CHAPTER}

Hepatocirrhosis is one of chronic diseases, and its causes are complicated. {END OF DISCRETE CHAPTER}

{BEGIN OF CLASSIFICATION CHAPTER}

According to the international standard, hepatocirrhoses are classified as micronodular cirrhosis, macronodular cirrhosis, mixed macro—and—micronodular cirrhosis, and incomplete septal cirrhosis:

(1). Micronodular cirrhosis: Microdular cirrhosis is also called Laennec's cirrhosis; microdular cirrhosis is usually caused by hepatosis viruses; the hepatosis viruses include hepatosis virus A, and hepatosis virus B; micronodular cirrhosis is mainly caused by hepatosis viruses, chronic alcoholism, malnutrition, and chemical poisoning;

If hepatosis viruses attack hepatosis, then hepatocytes necrosis occurs; on one hand, causing hepatofibres regeneration, causing the formation of portal hypertention, causing extravasated blood formed within the spleen, hence causing splenomegaly and animia; gradually, on the other hand, causing hepatic lubuli distorted, causing the hepatic blood circulatin disordered, thus, chronic inflammation occurs in hepatosis, and finally, hepatodynia and hepatomegaly appear!

(2). Macronodular cirrhosis:
...

(3). Mixed macro—and—micronodular cirrhosis:
...

(4). Incomplete septal cirrhosis:
...

{END OF CLASSIFICATION CHAPTER}

End.

{END OF PROGRAM}

References

[1] S. F. Smith, Adaptive Learning Systems, in Richard Forsyth(editor), Expert Systems, 168—189, Chapman and Hall(1984).
[2] R. C. Schank, Looking at Learning, in Luc Steels and J. A. Campbell(editors), Progress in Artificial Intelligence, 17—29, Ellis Horwood, Chichester(1985).
[3] Lu Ruqian, Enhance the Automatic Acquisition of Knowledge, N. 4, V. 4, JCST, 1989.
[4] Cao Cungen, Acquiring Knowledge From Domain Books, MS thesis, Institute of Mathematics, Academia Sinica, 1989.
[5] Wu Zhongbi, Pathology, People's Healthcare Press(China), 1987.
[6] Cui Qing, The research of Knowledge Acquisition, MS thesis, Institute of Mathematics, A-

cademia Sinica, 1987.

[7] Pan Zhiqing, Development of Expert Systems and Refinement of Knowledge Bases MS, thesis, Institute of Mathematics, Academia Sinica, 1987.

# Case-Oriented Knowledge Acquisition from Texts

Gabriele Schmidt & Franz Schmalhofer

German Research Center for Artificial Intelligence
University Bldg 57
Erwin-Schrödinger Str.
6750 Kaiserslautern

e-mail: schmidt@informatik.uni-kl.de

**Abstract.** For domains with different traces of expertise, integrative knowledge acquisition procedures which allow for an early knowledge validation are proposed. These procedures were developed within the model-based knowledge engineering framework. The advantages of such integrative knowledge acquisition and early knowledge validation are: The relevance of the elicited knowledge can be promptly assessed, inconsistencies discovered and knowledge gaps may be filled from another trace of expertise. In the current paper, knowledge acquisition from texts is enriched by utilizing records of solved cases as an additional trace of expertise. This enriched procedure is called case-oriented knowledge acquisition from texts. The proposed procedures are described and applied for the domain of mechanical engineering.

## 1. Introduction

Within the last decade, knowledge acquisition has emerged as an important area in expert system development. Knowledge acquisition is defined as the methodology for the formation of a model on expertise [4] so that this model can be used as a basis for a design specification (*design model*) and a subsequent top-down implementation of an expert or knowledge-based system. The view of knowledge acquisition as model building [16], has produced a number of principles and a systematicity on how such models can be formed. Problem classes [7], generic tasks [5] and interpretation models [4] assist in the formation of appropriate models on expertise.

For example, in Breuker and Wielinga´s KADS methodology the specification of a sequence of models is proposed. After analyzing some domain of expertise, an interpretation model which consists of an inference, task and strategic layer is selected [4]. Knowledge elicitation may then be supported through the selected interpretation model. From the elicited knowledge the so called domain layer is obtained, which together with the interpretation model forms the conceptual model. The conceptual model is thus the result of the *analysis phase* in KADS. From the conceptual model a *design* model for a future system can be developed which may then be used for a top-down implementation of the knowledge based system. In other words, in the *design and implementation phases* standard procedures from conventional software engineering can now be applied.

In model-based knowledge engineering, elaborate analyses are performed for a relatively complete specification of the future system. This approach is quite similar to conventional software engineering. Because knowledge-based systems are much more complex than conventional software and less well understood, the construction of a design model for a knowledge-based system is much more difficult than the specification for a conventional system. Therefore additional methods from cognitive science are required to build models of expertise from empirical data.

Contrary to rapid prototyping, which emphasizes early empirical tests of the functionality and the success of a future system [13], model-based knowledge engineering requires an extensive and time consuming *analysis phase*. As a pay-off of this time- and resource-investment into the analysis and design phases, it is expected that even more expensive complete re-developements of the system are avoided. In order to achieve this objective in an expert system development, it is suggested to perform an early validation of the elicited knowledge in the analysis phase. Through an early knowledge validation possible inconsistencies and irrelevant knowledge are eliminated and knowledge gaps are filled, so that a sufficient conceptual model (i.e. at least for the restricted task instances, which were considered in the analysis phase) is obtained. Such an early knowledge validation, which requires that several knowledge traces are available and integratively analyzed allows for the construction of a better design model.

In the following it is described how an early knowledge validation is performed in the area of mechanical engineering. At first a global analysis of mechanical engineering knowledge is performed. Historical, sociological and cognitive aspects of how mechanical engineering knowledge has developed are described. The resulting knowledge traces are identified and an interpretation model for production planning is selected. Based on the identified traces of expertise an integrated acquisition procedure is discussed, namely: the case-oriented knowledge acquisition method (COCAM) from texts.

## 2. Analysis of mechanical engineering knowledge

Our knowledge acquisition approach is a part of the ARC-TEC-project (ARC-TEC stands for Acquisition, Representation and Compilation of **TE**Chnical Knowledge) [17]. The application domain of the project and of knowledge acquisition is mechanical engineering. In mechanical engineering Computer Integrated Manufacturing (CIM) is an important objective. Computer Aided Planning (CAP) for the manufacturing of rotational parts is selected as example domain within CIM.

### *2.1. Analysis of HIstorical, SOciological and COgnitive aspects (HISOCO-analysis)*

The construction of a sufficient model on expertise (the conceptual model as well as the design model) depends on the selected interpretation model as well as on the applied elicitation method. Each elicitation method requires specific traces of expertise, from which knowledge can be extracted. For example, a successful application of a think-aloud method for knowledge elicitation may sometimes require that a top-level domain expert is accessible. Successful knowledge acquisition form texts on the other hand may require that qualitatively good publications or text books have already been written in a subject area.

A successful expert system development may thus be characterized through its global analysis of existing knowledge traces, possible interpretation models and appropriate elicitation methods. From such an analysis a coordinated set of knowledge traces, elicitation methods and interpretation models may be specified. A suitable selection can now be made, so that the knowledge traces required for the applied knowledge elicitation method are indeed available in the particular expert system project and that they fit with the interpretation model.

A global assessment of which knowledge traces are available in a subject domain in principle and whether these are accessible for the specific expert system project is therefore very useful. The domain knowledge of a subject area like mechanical engineering, which may change and develop over time, is often distributed among different organizations (e.g. companies, scientific community) and in the memories of different persons. The domain knowledge of a subject area is usually reflected in corresponding traces of expertise.

A developmental or historical assessment of a subject domain may show which traces of expertise have been established in an area. Thereby it may also be evaluated which changes may be expected in the near futures. A related analysis of sociological factors may show how knowledge traces are distributed in a subject area, which traces are publicly available and which traces are treated confidentially (e.g. within a company). Cognitive analyses of human expertise can show how additional knowledge traces may be obtained through probing or testing an expert's cognition.

Such a global analysis of the historical, sociological and cognitive aspects of expertise in a subject domain is termed HISOCO-Analysis.

Through unstructured interviews with a domain expert, which were spread out over a long time period, and a think-aloud study, the following HISOCO assumptions were made for mechanical engineering:

1) Mechanical engineering developed from a field of practitioners into an area where more and more scientific research is being performed (historical aspect).

2) The available knowledge is distributed among a number of specialists, who can be divided into practitioners and theorists. Whereas the theoreticians make their knowledge available through publications, the practitioners´ knowledge is often only stored in their memories and through their specific work products.

3) In mechanical engineering the practitioner´s normal problem solving method consists of adapting previous solutions to the newly arising situations. Through a cognitive assessment [21] such problem solving behavior can be characterized as case-based reasoning. The practitioner´s memories consist of a library of experiences or solved cases [18] rather than of a system of general rules.

From this analysis three traces of expertise were identified: 1) an implicit classification of cases exists in the practitioners´ memories, 2) records of previously solved cases are stored in the filing cabinets of companies; few of theses cases have been made more generally accessible through publications 3) theoretical knowledge about mechanical engineering is published in textbooks. A graphical representation of this result is shown in Figure 1.

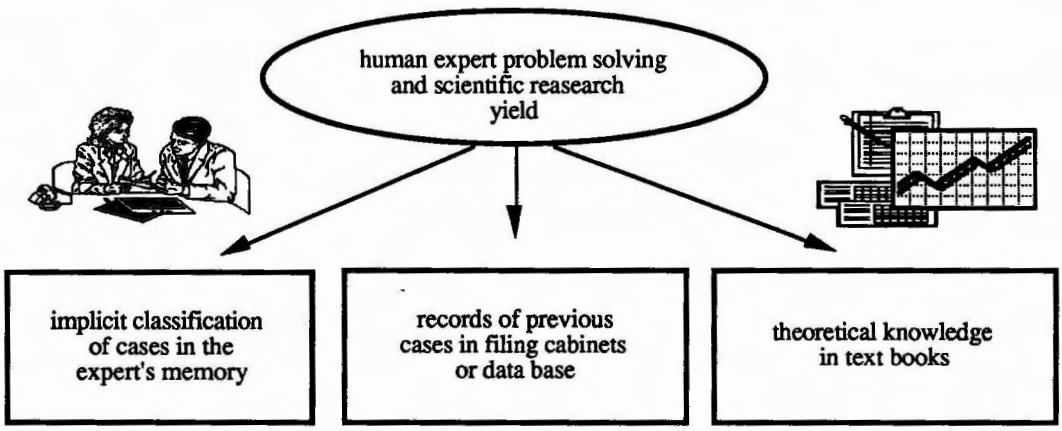



figure 1: Traces of expertise

*2.2. Selection of an interpretation model for production planning*

In order to make a first selection of an interpretation model, the expert´s problem solving behavior was investigated by a Think Aloud study [20]. The results of this study showed, that production planning is indeed a planning task - as the name already suggested. From this general classification and results from the area of cognitive modelling [18] as part of the HISOCO analysis, the interpretation model shown in figure 2 was established (compare to skeletal plan refinement [10; 16; 4]).

This interpretation model may be described as follows: In the beginning of the planning process the expert is presented the geometry and technology description of the workpiece, i.e. a drawing of the workpiece including geometry and technology data. The expert identifies a production class to which the workpiece belongs. Then a skeletal plan which is applicable for this class of workpieces is selected. Supposedly an expert´s memories include a hierarchy of workpiece classes. Results from cognition [6; 8; 11] provide substantial evidence that these classes have been formed according to skeletal production plans. The selected skeletal plan is then refined. The obtained plan is called principal plan. It is a full plan of the production process. It is consequently tested in a simulated execution, which may reveal some unforeseen problems. If such problems are encountered, the principal plan is modified or the workpiece is reclassified in order to select a new skeletal plan. This process is repeated until a satisfactory complete plan is obtained.

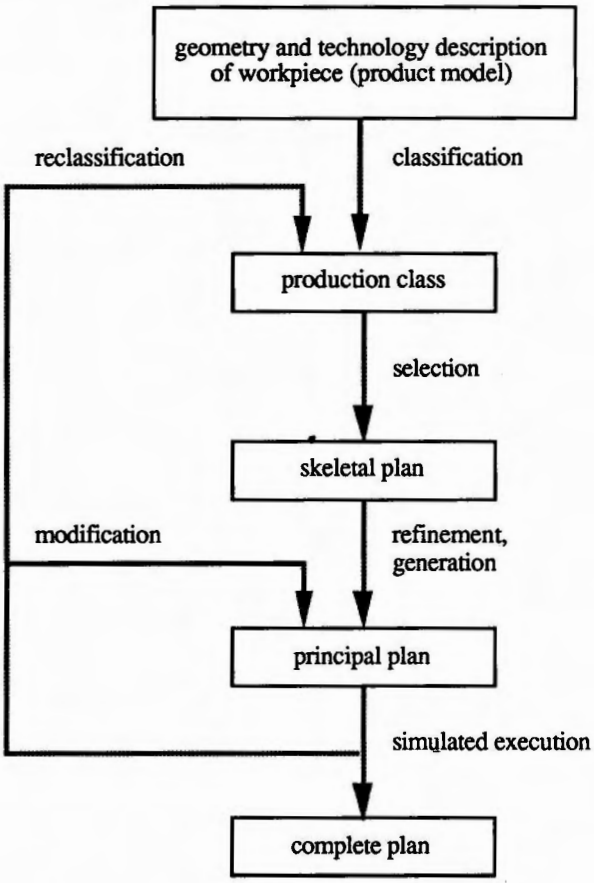



figure 2: interpretation model of planning process

The planning process described in figure 2 is called modification planning in mechanical engineering. Besides modification planning there are two other types of planning: replication planning and planning from scratch. Replication planning is a special case of modification planning, where a complete plan can be selected at once, the planning process is called replication planning. On the other hand sometimes no appropriate skeletal plan can be selected. Then planning from scratch by using general rules has to be done. But most production planning is modification

planning. This type of planning corresponds to human thinking and is recommended by textbooks of the domain [9].

Since modification planning is the most typical type of planning which is also confirmed by experts and textbooks, we have selected an interpretation model based on modification planning, i.e. skeletal plan refinement. The task structure of skeletal plan refinement may be decomposed into functions like select a skeletal plan, refine the skeletal plan and validate the refined plan.

In order to construct a conceptual model from the selected interpretation model different types of knowledge must be acquired on the domain layer, such as knowledge which is needed for the classification of product models, skeletal plans which are associated with product models and knowledge which is used in the refinement of skeletal plans. As the HISOCO-analysis shows there are three different traces of expertise from which knowledge for production planning by skeletal plan refinement can be required. Knowledge about the class hierarchy has to be acquired from the experts (see [19])whereas knowledge for skeletal plan refinement can be found in textbooks. The textbooks knowledge should however validated as to whether it can really be applied to individual cases. Therefore a procedure for case-oriented knowledge acquisition (COCAM) from texts was developed and is described in the following section.

## 3. Case-oriented knowledge acquisition from texts

### *3.1. Basic ideas of case-oriented knowledge acquisition*

The knowledge written down in textbooks consists mainly of general rules. As pointed out in the previous section, these rules are the summarized results of previous problem solving efforts. They are based on a large number of individual cases, and it is usually expected that they apply to all cases. If the rules found in textbooks are indeed correct and furthermore complete for the solution of the to be solved problem, knowledge acquisition is a simple task and can be performed without the help of individual cases.

In a practical domain such as mechanical engineering, the rules found in textbooks are rarely correct in a strict logical sense and they are far from being complete. Furthermore, many textbook rules are rather vague statements for which sometimes even different interpretations are offered in the same text. When the expert is asked in such a situation, he will usually say ´it depends´ and name a number of individual cases.

Since case-oriented thinking seems to be the expert´s natural way of thinking, it is suggested to acquire knowledge from a text with the help of solutions to individual cases. The expert is asked to give justifications for the solutions to the individual cases in terms of the rules of the selected text. When rules of the text are insufficient, the expert is to name rules from other texts or from his memory. This procedure guarantees that the acquired knowledge is necessary and sufficient at least for the solution of the selected cases. Of course, the acquired knowledge must be tested with different cases, but this can be performed partially automatic. The advantages of case-oriented knowledge acquisition from texts are that the relevance and the completeness of the acquired knowledge (for a limited number of cases) are tested at an early stage of knowledge acquisition.

### *3.2. Procedure for case-oriented knowledge acquisition from texts*

There are three conditions which have to be satisfied in order to apply the procedure for case-oriented knowledge acquisition from texts:

- There has to be theoretical knowledge, i.e. textbooks of the respective domain have to exist.
- At least one expert has to be found, who takes part in the inquiry.
- There have to be example tasks available, which can be applied without too much effort.

In order to apply this procedure there must exist theoretical knowledge which is written down in textbooks, catalogues etc.. In some domains this demand may be satisfied for one subtask of the domain and violated for another subtask. Therefore, every time knowledge for a new subtask is to

be acquired it must be checked whether the respective theoretical knowledge exists.

Since the knowledge engineer is not an expert in the to be acquired domain and is not intended to turn into one [12; 3], he cannot decide which knowledge contained in a textbook may be relevant for solving the subtask. He therefore needs the assistance of some domain experts.

Since experts usually have acquired their basic knowledge from textbooks they should be able to select text-passages which are relevant for solving a specific problem. Practicers, however, may have difficulties in estimating the relevance of text passages, since they are not confronted with this task in their daily work. In practice practicers solve a concrete problem by reasoning about the specific conditions and specific consequences of an action (see [2]) They do not reason about the general rule. For that reason after selecting text-passages which contain general rules they can validate the rules by explaining particular cases. When particular cases are explained, knowledge gaps can be discovered which must be filled. Therefore the previously recorded cases are of crucial importance for case-oriented knowledge acquisition and beside the theoretical knowledge in books practicable examples are needed, too. Sometimes it may not be easy to find detailed and appropriate examples.

If the three conditions are satisfied, the interpretation model and the analysis of historical, sociological and cognitive aspects (HISOCO) enable the knowledge engineer to apply the procedure and thereby give instructions to the expert.

In order to apply the procedure, some knowledge about the task structure is helpful but not necessary. Such knowledge can be gathered for instance by Think-Aloud protocols. Think-Aloud protocols provide a first rough decomposition of the domain-specific task into subtasks. For each subtask the procedure can be applied independently. The procedure can be executed with paper and pencil or with the support of a tool (which has been specified but not yet implemented).

The procedure which will now be described can be roughly structured in three phases:

phase I: Selection of texts and cases

phase II: Application of the text segments to the cases and filling of discovered gaps

phase III: Analysis of acquired knowledge

During the first phase the materials which are used for knowledge acquisition, i.e. texts and cases are selected. The most relevant text segments are marked in order to be more easily accessible in the second phase of the procedure. This selection of text segments is similar to usual methods of knowledge acquisition from texts. In the second phase the marked text segments are applied by the expert to explain the selected cases. Thereby gaps in the selected textbook knowledge are identified which are filled from other text passages or common sense knowledge. In the last phase, the acquired knowledge is interpreted and analyzed.

Phase I:

First the knowledge engineer tells the expert for which subtask of the domain the knowledge is to be acquired. The expert has to select the appropriate domain-specific literature, whereby the total number of textbooks is not limited. If too many textbooks are considered relevant by the expert, the expert is requested to partition the subtask in order to make the knowledge acquisition process more manageable.

The expert identifies and marks all the text-passages he considers as relevant for the solution of the selected task. The length of each marked passage can range from single sentences to whole paragraphs.

Then the knowledge engineer has to find an appropriate example and its solution. Since this task can overcharge the competence of the knowledge engineer, it can be performed with the assistance of other domain experts. It goes without saying that the expert performing the following phase should not know the specific examples in advance.

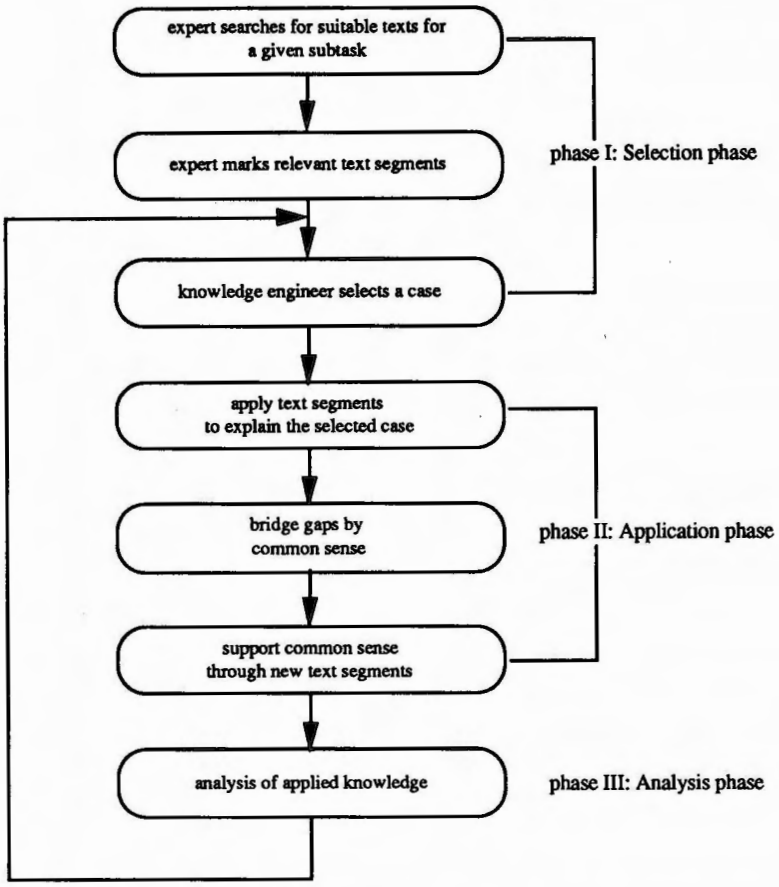



figure 3: procedure for case-oriented knowledge acquisition

Phase II:

The expert tries to explain the solution of the given example using the marked passages. Thereby not every marked sentence will be utilized and some yet unmarked passages may be missing. In order to bridge the knowledge gaps, he is allowed to use his own explanations (common sense). By this way a full explanation of the solution is obtained.

Then the expert and the knowledge engineer identify unmarked text-passages in the textbooks which support the expert´s explanations. If no such text-passages can be found there exist gaps in the theoretical knowledge of the domain. These gaps must be filled from other knowledge sources such as the expert`s episodic knowledge or technical common sense. This knowledge can be validated by repeating the procedure with the same example and the same text-passages but with other experts.

Phase III:

The knowledge which was acquired in phase II can be classified as belonging to one of the following three classes: marked text segments which were used in the explanation of the particular example, text segments which were not used and additional knowledge which was used to fill the knowledge gaps. The latter consists of knowledge from additional text segments which had not been selected being relevant in phase I, of episodic knowledge and of technical common sense.

For the knowledge in the first class it can be analyzed in which sequence the

individual text segments were use by the expert. These segments can be formally represented as rules and the whole explanation can thus be transformed into a proof tree for the selected case. The proof trees obtained for different examples can be compared to each other and the expert´s inference structures can thus identified.

The knowledge which was marked as relevant but was not used can be checked wether is was not applicable in principle, wether it was overlooked by the expert or wether it was considered inappropriate. Knowledge elements which were found to be not applicable for large number of cases need not be included in the knowledge base. The overlooked textbook segments should be presented to the expert and he should reconsider his original explanations. Reasons should be given by the expert why he considered principally applicable knowledge has been inappropriate. These reasons can be used as discriminating conditions for the application of the individual rules.

Based on the knowledge analysis a new case is selected which has to be explained by the expert using the same marked text segments. Thereby it will be determined if the same or different rules are used. Figure 3 shows the iterative application of the procedure for different cases.

From the common sense and episodic knowledge which was identified requirements for other knowledge acquisition procedures can be derived. Since the represented procedure has only designed for case-oriented knowledge acquisition from texts is must be supported by other procedures which are more appropriate to elicit other traces of expertise, i.e. episodic knowledge.

### *3.3. Results of an application of the procedure*

In the ARC-TEC-project the procedure described above is practicable because the three preconditions are satisfied. Mechanical engineering has a theoretical foundation and there are enough textbooks available. One domain expert is in the project who can take part in knowledge acquisition procedures. Previously recorded cases, i.e. product models and production plans, can be obtained from catalogues or archives of producers.

The procedure is applied for the acquisition of the knowledge which is needed to construct a production plan for rotational parts. Since the construction of a production plan is a rather complex task it should be subdivided into some subtasks. Based on Think-Aloud protocols production planning for rotational parts (´Drehen´) can be decomposed into the three steps as shown in figure 4: selection of chucking (´Aufspannung´), selection of a cutting sequence (´Schnittaufteilung´) and selection of tools (´Werkzeugauswahl´). Since the first step is rather independent of the subsequent steps the procedure was first applied for the acquisition of knowledge about chucking. The procedure can be applied for the other steps as well, if the results of the preceding steps are available.




figure 4: subtasks for production planning

The case-oriented knowledge acquisition procedure was performed with two domain experts and two different examples. One expert had written a text about the selection of chucking tools (theoretical expert) whereas the other expert had a lot of practical experience (practitioner). We will now describe the experiences made and the results obtained through the application of the procedure.

Phase I:

Both experts chose the same text-passage of 15 pages[1]. The theoretical expert marked 17 and the other expert 36 sentences or formulae as relevant. In order to take into account the different kinds of chucking tools the knowledge engineer selected two examples with different chucking tools from a catalogue.

Phase II:

The experts explained the two examples. The theoretical expert used a total of nine text segments in the two explanation. Moreover three knowledge gaps were discovered, i.e. the expert has used three rules which could not be found in the marked text. Two of these rules could be found in other text-passages of the same textbook. For the third knowledge gap no adequate text-passage could be found. However, the expert was able to provide an appropriate common sense explanation.

The practitioner used six of the 32 marked sentences. Five knowledge gaps were discovered one of which could again not be covered by theoretical knowledge.

Phase III:

The knowledge analysis has not yet been completed. Since only a small portion of the marked text segments were used for the explanation of the two examples, different cases will be selected for which the unmarked passages seem to be relevant.

The conjecture that the practitioner has difficulties in estimating the relevance of textbook segments is reinforced by the observation that the practitioner used only 10 of the 36 marked text segments whereas the theoretician used 12 of 17 marked text segments.

By applying the method an important fact was discovered: For a specific problem, the theoretical knowledge often contains more than one applicable statement. However, it does not provide the possibility to decide, which statement will be best used in a specific case. Furthermore the theoretical knowledge is sometimes not specific enough to be applied to a real domain problem. Therefore the theoretical knowledge must be supplemented by technical common sense knowledge. Such knowledge can be acquired and validated by offering the same task and example to other experts during application of our method.

## 4. Discussion

There are a number of similarities between the proposed knowledge acquisition procedure and documented methods of the state of the art [3]. In particular, a close relation exists to knowledge acquisition tools, which combine different knowledge eliciting methods, like repertory grids, text and interview analysis [14] and to different knowledge acquisition methods from texts [22; 15].

KRITON (see [14]) is a system that consists of three tools, one used for protocol analysis, one used for interview analysis and a tool for text analysis.

The case-oriented knowledge acquisition method from texts resembles combined knowledge elicitation tools, such as KRITON in that several knowledge traces can be analyzed. The major differences exist in the following observations: Whereas COKAM provides a clearly defined procedure for combining the different knowledge sources, KRITON is open to all possible combinations. On the other hand, COKAM is restricted to domains, where well thought through cases with their solutions as well as theoretical texts which relate to the cases are available.

The system of [15] for knowledge acquisition from texts consists of two parts PROPOS and EPISTOS. The task of PROPOS is the systematic and semi-automated encoding of natural language texts into a suggested representation. There are meaningful categories, which support the knowledge engineer to select relevant knowledge elements and to classify them. This form of

[1]The source is a "Studienarbeit an der Universität Kaiserslautern, Fachbereich Fertigungstechnik und Betriebsorganisation," written by Ottmar Ernst

support is based on the linguistic aspects. EPISTOS selects knowledge elements and classifies them in regard to a model of the domain.

The system of [Szpakowicz 1988].supports semi-automated acquisition of structures from technical texts. It presumes an initial general model according to the KADS-method and then acquires conceptual structures from texts. The operator supports the semi-automated system if it cannot take decisions on its own.

The KADS power tools [1] provide a hypertext-based protocol and concept editor, where the words of the stored text are mouse-sensitive, so that the definitions of the words can be inspected. Also, annotations are stored so that remarks from different people can be read on demand.

The implementation of the COKAM tool has only been started. Consequently, it does not yet provide the variety of different services of these other tools. COKAM differs from these tools in that texts are analyzed with respect to specific cases. Thereby, the relevance of text information for a specific subtask can be assessed early on in knowledge acquisition. The benefits of such an early knowledge validation can be already substantiated by the very preliminary application of the method which was described in this paper. With COKAM contradictory rules and too vague rules which need specification by common sense knowledge can be detected at an early stage of knowledge acquisition.

## 5. Acknowledgements

We wish to thank Otto Kühn for his critical remarks to earlier versions of this paper. We would like to acknowledge the assistance of Ottmar Ernst, Gernot Laufkötter, Ralf Legleitner, Beate Schlenker, and Athanasios Valus in this research.

## 6. References

[1] Anjewierden, A. *The KADS System* Proceedings of European Knowledge Acquisition Workshop (EKAW´87), Reading, (1987),

[2] Becker, B. *Towards a concept for case-based knowledge-acquisition.* Proceedings of European Knowledge Acquisition Workshop (EKAW´88), Bonn, (1988), p. 9-1 -9-12

[3] Boose, J.H. & Gaines, B.R. *Knowledge Acquisition for Knowledge-Based Systems: Notes on the State-of-the-Art* Machine Learning,5,377-394, (1989)

[4] Breuker, J. & Wielinga, B. *Models of expertise in knowledge acquisition.* In Guida & Tasso (eds.): *Topics in Expert System Design: Methodologies and Tools.* North-Holland, Amsterdam (1989)

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

[6] Chi, M., Feltovich, P. & Glaser, R. *Categorization and representation of physics problems by experts and novices.* Cognitive Science, 5, (1981) 121-152

[7] Clancey, W. Heuristic Classification. Artificial Intelligence 27 (1985), 289-450.

[8] Egan, D. E. & Schwartz, B. J.: Chunking in recall of symbolic drawings. Memory and Cognition 7 ( 1979) 149-158.

[9] Eversheim, W. *Organisation in der Produktionstechnik, Band4, Arbeitsvorbereitung.* VDI Verlag, Düsseldorf, (1989), p. 125 - 128

[10] Friedland, P.& Yumiiwasaki The Concept and Implementation of Skeletal Plans, Journal of Automatic Reasoning,11085, pp 161 - 208

[11] Glaser, R. & Chi, M.T. Overview. In: Chi, M. T., Glaser, R. & Farr, M.J. (eds) The Nature of Expertise. Lawrence Erlbaum Associates, Hillsdale, N.J. 1988

[12] Karbach, W. *Methoden und Techniken des Knowledge Engineering*. Arbeitspapiere der GMD 448, p. 45 (1988).

[13] Laske, O.E. Ungelöste Probleme bei der Wissensakquisition für wissensbasierte Systeme. KI 4/89, 4-12.

[14] Linster, M. *A Critical Look at Kriton: Moving from first to second generation.* 4rd AAAI-Sponsered Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, (1988) p. 17-1 - 17-19

[15] Möller, J.-U. *Knowledge Acquisition from Texts*. Proceedings of European Knowledge Acquisition Workshop (EKAW'88), Bonn, (1988), p. 25-1 -25-16

[16] Musen, M. Fagan, L., Combs, D. and Shortliffe, E. *Use of a domain model to drive an interactive knowledge-edition tool.* Int. Journ. of Man-Machine Studies 26 (1987), p. 115-121

[17] Richter, M., Boley, H., Wetter, Th. & Warnecke, G. ARC-TEC: Acquisition, Representation and Compilation of Technical Knowledge, Projektantrag, German Research Center for Artificial Intelligence, Postfach 2080, D-6750 Kaiserslautern,1989

[18] Riesbeck, C. K. & Schank, R.C. (1989) Inside case-based reasoning. Hillsdale, N.J. Lawrence Erlbaum

[19] Schmalhofer, F. & Bergmann, R. *Extending Expert Problem Solutions for Technical Planning Tasks*. to appear: Proceedings of the 2. Knowledge Engineering Workshop, Berlin (1990)

[20] Schmidt, G., Legleitner, R. & Schmalhofer, F. *Lautes Denken bei der Erstellung der Schnittaufteilung, der Werkzeugauswahl und Festlegung der Maschineneinstelldaten.* Interner Bericht des ARC-TEC Projektes (1990)

[21] Strube, G. & Janetzko, D. (im Druck) Episodisches Wissen und fallbasiertes Schließen: Aufgaben für Wissensdiagnostik und Wissenspsychologie. Schweizerische Zeitschrift für Psychologie

[22] Szpakowicz. St.. *Semi-Automatic Acquisition of Conceptual Structure from Technical Texts*. Proceedings of 4rd AAAI-Sponsered Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, November 1988

# Knowledge based knowledge acquisition: the next generation of support tools

Nigel Shadbolt
Artificial Intelligence Group
Department of Psychology
University of Nottingham
Nottingham NG7 2RD
England

Bob Wielinga
Social Science Informatics
University of Amsterdam
Herengracht 196
1016 BS Amsterdam
The Netherlands

**Abstract**

In this paper we present a framweork for knowledge based knowledge acquisition. After reviewing the various classes of knowledge engineering support tools that are available, we discuss our approach to the process of knowledge acquisition and its role in the KBS lifecycle, We then describe an integrated knowledge engineering workbench (KEW). KEW supports the knowledge engineer in the process of knowledge acquisition. KEW brings together a range of knowledge acquisition techniques, including machine learning methods. The aim is to make KEW an active and directive system. This requires that KEW has knowledge about knowledge acquisition. We describe via a detailed set of episodes and examples the kinds of knowledge that animates KEW. This knowledge is shown to include; knowledge about how to conduct KA, how to use particular tools, how to transform between then, how to integrate their results into a consistent and evolving application KBS.

## 1 Introduction

The ACKnowledge[1] project addresses the problem of supporting the knowledge engineer in the phases of KBS development involving knowledge acquisition. There has been a lot of work that has resulted, either intentionally or unintentionally, in potential support for this process. This work includes the development of AI programming environments, specific knowledge elicitation tools, machine learning techniques, and knowledge acquisition methodologies.

Despite this research substantial problems remain with the acquisition of knowledge from human domain experts, cases, performance records and documents. One reason for this is that the available tools and methods have not been brought together within a single integrated and principled system.

[1]This project is funded under the auspices of the ESPRIT programme (Project No 2576). The project consortium consists of the following institutions; Cap Sesa Innovation, Marconi Command and Control Systems, GEC-Marconi Research Centre, Telefonica, Computas Expert Systems, Veritas Research, the University of Amsterdam, Sintef, and the University of Nottingham

The goal of ACKnowledge is to provide just such a system or knowledge engineering workbench (KEW). Within this project we have been working on the problem of providing a coherent framework for the integration of methods and techniques within KEW. In particular, we have adopted the view that KEW will act as a KBS for KA. This view has a number of consequences for the development of what we conceive to be the next generation of knowledge acquisition tools. These issues will be explored in this paper. The most important is the realisation that such a KA KBS will have to be an active, dynamic system animated by knowledge about how to conduct KA, how to use particular tools, how to transform between them, how to integrate their results into a consistent and evolving application KBS.

In the next section we review attempts to date to support the KA process with software tools. We then describe how work in the ACKnowledge project is positioned with respect to existing KA support tools. The fundamental assumptions we make about the KA process and KEW's role within it are outlined. We then provide an extended description of how we see KEW operating as a knowledge based KA system, we discuss the classes of knowledge that are required to implement this approach.

# 2 Support tools for KA

A number of approaches to automated support of the KA process have been developed. These software tools fall into three categories: (i) those which have been developed for specific domains (domain dependent); (ii) those which are computer implementations of particular KA techniques (domain independent); (iii) and those which are *integrated systems* for acquisition support.

Domain specific tools are tailored to elicit the types of knowledge known to be important in a particular domain. The SALT system [Marcus *et al.*, 1985] has in-built knowledge of the problem solving strategies used when configuring electro-mechanical systems. The expert interacts through an automated interview which allows the selection by menu of elements in the domain. The interview results are automatically converted into rules, and the expert has an opportunity to edit the resultant rules. A similar system, KNACK [Klinker *et al.*, 1987] has been used to construct expert systems in the domain of nuclear hardening. A slightly different approach is taken by [Friedland, 1979]. In his system, expert geneticists code the knowledge base themselves in a language specifically tailored to genetics. These systems are by definition restricted to particular domains. However, it is alleged that they can be adapted for use in domains which share similar structures, and problem solving strategies.

There are also tools which implement specific acquisition techniques. For example, a natural method to automate is the repertory grid since manual elicitation and analysis is time consuming but quite straight-forward. To this end [Boose, 1985] developed ETS (Expertise Transfer System). This system supports interactive elicitation of a grid from the user. The completed grid is used to construct implications. This system has been used on a very great number of domains (see [Boose, 1986] for a list) and delivers fast prototype systems.

A number of machine learning (ML) methods have been implemented which can also serve as KA techniques. Some of the most common are the similarity-based learning methods (or induction). In SBL, examples and counter examples of a concept are used as a basis for deriving a description of the concept. Thus Michalskis' AQ11 algorithm induces rules in the form of structural descriptions that discriminate positive and negative examples of a

concept.

Experience with these ML techniques has shown that they require quite large numbers of examples to produce correct generalizations, that they are sensitive to errors in the data, and the particular nature of the description of the case.

Much knowledge engineering is still done by hand: knowledge engineers document their understanding of the domain and then build a system, using a conventional programming environment, an AI programming system or an expert system shell. Structured editors, using graphical interfaces are useful tools in this process. They can be used to produce written documentation (including diagrams) or executable knowledge bases. Two state of the art systems are KADS Power Tools [Anjewierden, 1987], a system that produces structured written documentation, and KEATS [Motta *et al.*, 1989].

KADS Power Tools provides bottom-up support for knowledge acquisition by offering a set of tools, including a protocol editor and a concept editor. The protocol editor supports the identification of knowledge at the linguistic level using a dedicated hypertext system [Nelson, 1980]. The concept editor supports the organisation of knowledge at a conceptual level. The concept editor supports the definition of individual concepts, their attributes and values, and the interelations between them. Both of these tools produce documentation - formatting and printing information edited in the KADS system.

What is generally seen to be desirable is a move towards the integration of actual elicitation and acquisition techniques. We find a simple kind of integration in LEX2 [Mitchell, 1983]. LEX2 can analyse its own knowledge base to identify examples that it cannot reliably classify. These examples can be presented to a problem solver (human or machine) in order to indicate the need for new training examples.

A system called BLIP [Morik, 1987] has the user enter various kinds of information; general facts and rules, specific facts, metalevel descriptions of facts and inference rules. The system checks the effects of modifications of the knowledge base, looking for inconsistencies, redundancies and interesting patterns. These are then reported back to the user. The dialogue can be controlled by the user or the system. Part of BLIP is an inductive learning system that attempts to infer general rules from its domain knowledge, guided by the metalevel descriptions. The integration of induction with direct editing helps the user to structure the knowledge base and to keep track of the implications of the knowledge entered into the system.

A number of systems drive computer based elicitation in a model based fashion - relying on *compiled* views of the plans/structure of a domain. The most relevant example of this method is ROGET [Bennet, 1983] in which the user is asked a large number of questions about domain concepts and domain relations. ROGET is equipped with a model of diagnostic reasoning found in MYCIN, and the dialogue is driven by this model. In OPAL [Musen *et al.*, 1988], a knowledge acquisition system for ONCOCIN [Shortliffe *et al.*, 1981], a strong domain model is used to generate forms for the user to fill in. In this way the user fills in the knowledge base for a specific application. These systems can be seen as slight generalisations of the domain specific elicitation tools discussed earlier.

The emphasis of the ACKnowledge project is on the integration of a much wider variety of techniques into a coherent system. Since many of the available tools differ in the way in which they can support KBS development, KEW will have to be based on a synthesis of the various approaches that have been explored in the past. The approach that currently seems

most promising is an active support system where the system uses knowledge that is entered by the knowledge engineer, obtained by elicitation, induction or other means and uses these partial knowledge bases to create an adequate knowledge base for the final KBS.

Important resources for the knowledge engineer who will use KEW are a library of generic models and an explicit representation of ongoing and planned activities. Using the same resources, KEW can even play an active role in the KBS development process, but it will have to rely on the human user in those tasks that are beyond the current state-of-the-art. In what follows we will lay the foundations for such a system.

# 3 Fundamental assumptions

In the following sections we outline our *philosophical* orientation in respect of the nature and role of KA in KBS construction.

## 3.1 The role of models in KA

Currently, a number of the most influential approaches to knowledge acquisition conceive of the process as essentially a modelling activity [Morik, 1989]; [Breuker & Wielinga, 1989]. The use of models is a means of coping with the complexity of the development process. A model reflects, through abstraction of detail, selected characteristics of the object, device or process in the real world that it stands for [DeMarco, 1982].

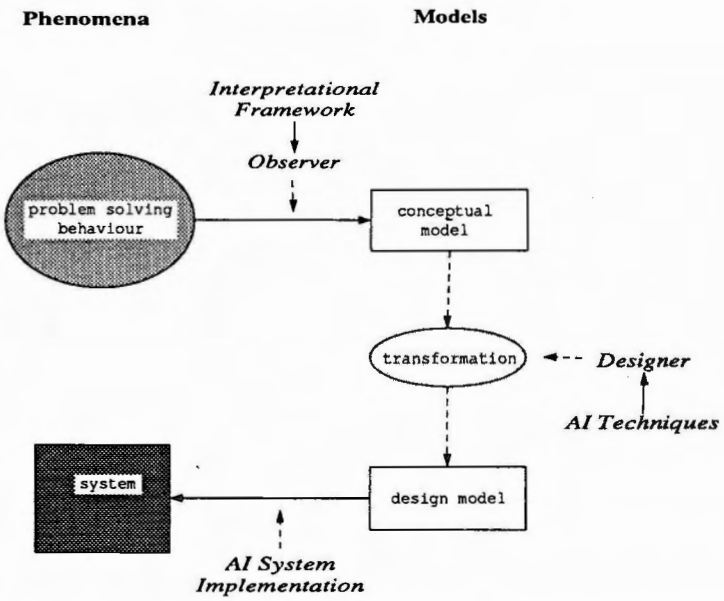



Figure 1: The KBS Development Process: A Bird's Eye View

Fig. 1 symbolises our view of the knowledge engineering process. An observer (knowledge engineer) constructs a conceptual model by abstracting from the problem solving behaviour

of experts. This abstraction process is aided by the use of an interpretational framework, such as generic models of tasks or of task domains. The conceptual model is a knowledge-level [Newell, 1982] model of the domain expertise. The conceptual model is oriented towards the real-world in the sense that it is phrased in real-world terminology and can be used as a means of communication between the knowledge engineer and expert. The conceptual model does not take into account detailed constraints with regard to the system under construction (the artifact).

The *design model* is a model of the artifact. It describes how the conceptual model is realised with particular AI techniques and methods. The idea is that the design model can be transformed into a detailed system design and subsequently into actual *system code* without further major design decisions having to be made.

Directing knowledge engineering through the use of conceptual models is a central feature of our work. We distinguish four types of knowledge that can be discriminated at the conceptual level and which can go to form the content of such models. Firstly, *domain* knowledge. This is knowledge that *describes* the concepts and elements in the domain and relations between them. A second type is *task* knowledge. This is knowledge to do with how goals and sub-goals, tasks and sub-tasks should be performed. It also characterises the input and output features of data to these tasks. A third type of expert knowledge can be referred to as *strategic* knowledge. This is information that monitors and controls problem solving. This can have to do with the way resources are used. What to do if the proposed solution fails or is found to be inappropriate in some way. What to do when faced with incomplete or insufficient data. Finally there is the *inference level.* This is knowledge about how the components of expertise are to be organised and used in the overall system. The inference level is sometimes used to discriminate between different generic classes of expert system.

Given these different kinds of conceptual knowledge it is perhaps not surprising that, as we have seen in the review of existing support tools, a number of model driven approaches abound. Thus KADS principally uses on-line libraries consisting of models at the inference level to direct acquisition. These models help the knowledge engineer to structure the information that is available about the expertise and which is to be incorporated in the KBS. An alternative approach in many ways similar to KADS makes use of task level representations [Chandrasekaran, 1987], [Bylander & Chandrasekaran, 1988], [Clancey, 1985]. The approach is to use general, invariant features of a task to guide the acquisition by a system of detailed knowledge. Other work, as we have seen, uses strong domain specific representations. These are used as a language for implementing expert systems (see OPAL, [Musen *et al.*, 1988]). A number of researchers exploit the fact that many KBS applications are built in similar, if not nearly identical domains [Eshelman, 1989], [Marcus *et al.*, 1985].

A key research issue is how these various model driven approaches can be integrated. And how does one select the most appropriate model driven methods for any application.

## 3.2 The nature of the Knowledge Acquisition cycle

We view knowledge acquisition as having a cyclic structure: data are collected, interpreted, integrated with existing information, internally stored, and plans for a new cycle are made. This cycle occurs in different phases of the life cycle and hence forms part of most activities in knowledge acquisition. However the nature and emphasis of the data collection, the interpretation and integration processes vary with both the particular activity in which they

occur and the model that is modified (or created) by them. In particular, we distinguish the following activities: planning a KA sequence of operations, technique selection, technique application, assimilation of the results, evaluation of the revised KB.

We can summarise the cyclic, iterative nature of the KA process in figure 2

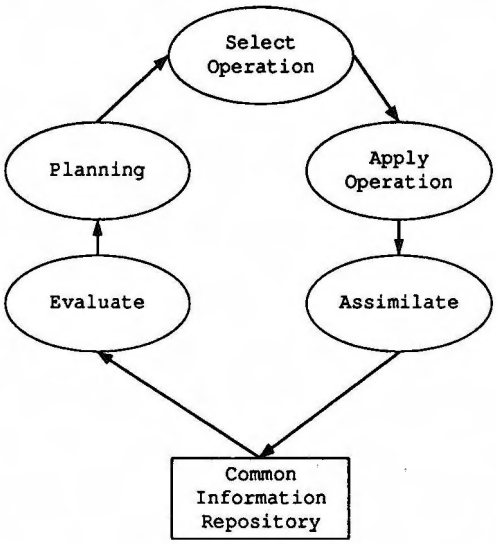



Figure 2: The Knowledge Acquisition Cycle

## 3.3 Functions of KEW

If we consider the various proposals for supporting KA we can see that together they specify a need to provide four *levels* of support activity in KEW.

- Clerical activities. One kind of support that KEW should provide is for user activities, such as protocol browsing, editing of examples for various KA tools etc. Many of these are essentially clerical support functions. They are nevertheless important.
- KA tool execution. We also require KEW to be able to run a range of KA tools and techniques. This in turn requires the gathering of data, execution of the technique and presentation of the results.
- Perform and support operations on the results of KA tools. A fundamental feature of KEW's activity will be executing processes of knowledge transformation and integration both within and between tools. To perform these operations, and to understand the consequences of performing them, we require knowledge. For instance, which transformations are inference preserving, which result in information loss and so on.
- Provide context dependent advice and guidance. At the heart of KEW we must have knowledge sufficient to plan, suggest and execute sequences of KA methods. We need to know what techniques to use, when to stop using certain methods etc.

As we progress through our levels of support, KEW effectively becomes more active and directive. The corollary is that it requires more and more substantive knowledge to support these kinds of activity.

## 3.4 The Animating Knowledge in KEW

What sorts of knowledge will KEW be able to call on to support these various activities? We can distinguish at least four major types of knowledge that could provide a foundation for an active KEW.

The first of these is knowledge about the KA process. This amounts to the encapsulation of expertise to do with conducting KA - advice and guidance about what to do and when. This would need to be context sensitive with respect to the the activity cycle postulated in Figure 2. Such advice and guidance could take many forms but would include; how to plan a sequence of KA activities, what are the cost benefits of such a plan [Burton *et al.*, 1987], [Burton *et al.*, 1988], [Shadbolt & Burton, 1989], what domain/application characteristics constrain particular kinds of KA activity.

This type of knowledge is likely to present itself in KEW as advice and guidance rather than absolute control over the KA activity.

The second source of knowledge has to do with *generic models* in the broadest sense of the term. We mean by this the type of generic knowledge extracted and formulated into models of problem solving methods, epistemological categories, domain structures and content. This type of knowledge will play an important role in KEW - directing the course of KA.

Thus the adoption of a model of diagnosis via heuristic classification allows us to exercise control on the use of tools, the structuring of the KB, and the control of the target KB. Directing acquisition via templates that specify the kinds of structural and static knowledge that may be present in an application or domain is another example of the use of model knowledge. This sort of expectation driven acquisition could also make use of the general character of knowledge. For example, whether data is measurable, certain, or polymorphic may be knowledge that can direct the use of a particular acquisition tool. These various points are exemplified in the following sections.

Knowledge about the KA tools themselves constitutes a third influence that might direct KEW activity. Thus repertory grids make assumptions about data, how it is analysed and what the status of the results is. The knowledge engineer should be made aware of, or in some other manner directed through, these issues. Background processes can be envisaged that examine the data used and produced by a tool. Such processes could offer suggestions or critiques about the nature of results and their possible interpretations. Returning to our repertory grid example KEW may seek to determine whether the dimensions along which elements are to be ranked are ordinal, metric, euclidean etc. Are the elements themselves homogeneous. Such knowledge can be used in very direct ways. A common error in repertory grids is to fail to appreciate that many analyses are rendered meaningless if one dimension in the grid is treated as a linear scale and another as logarithmic.

Finally we can imagine that knowledge gathered through continuous reasoning by KEW on the results of KA activity could also be important. We might refer to this as *evaluative* knowledge. KEW would contain knowledge about the analysis of the knowledge accrued.

Examples include simple KB analyses for consistency, coverage, generalisation and specialisation.

Each of these types of knowledge is important in the vision of KEW which we present. It is a vision that tries to illustrate how the various active aspects of KEW could be made to work in practice.

# 4 Getting started: task elicitation

The types of knowledge discussed in the previous sections characterised the classes of knowledge that might inform a KBS for KA. In what follows we try to show how these categories of knowledge might figure in an active KBS KA system. We detail some of the actual declarative content of this type of knowledge. The knowledge that animates our proposed system is knowledge about; the KA techniques themselves, knowledge engineering, knowledge about the generic structure of problems and tasks, knowledge the system has about its own architecture and processing.

To develop our ideas we prototyped KEW using a range of KA techniques and an actual existing application. The application XFrac was developed by one of the consortium partners - Computas Expert Systems. XFrac analyses fractures of steel components. The ACKnowledge project had assembled various KA materials used in the original construction of XFrac; protocols, concept lexicons, interviews, cases, intermediate and final rule bases. What we have done is attempt to recapitulate the construction of the original system, originally by hand, using the KBS KA system.

The task we set ourselves was to imagine how our proposed system would proceed to construct a KBS in conjunction with our knowledge engineer, and domain expert.

We did this through a series of episodes representing interactions with KEW, in most cases we used implemented KA tools and real KE data. For ease of exposition we have formulated a number of episodes in terms of natural language interactions. The surface realisation of these episodes is incidental. It is the intentions and goals behind these episodes that are important and that we have, to varying degrees, operationalised in our prototypes.

We assume a situation where at the outset of KA the application knowledge base is empty. In the absence of any knowledge about the application KEW must attempt to characterise the task. KEW's knowledge acquisition knowledge bases contains relatively simple rules that furnish KEW with an initial set of goals that set subsequent KA in motion.

These rules set up an agenda of goals for the KEW to pursue. They are essentially control knowledge of the category 'knowledge about conducting KA', see for example table 1. The various goals will in turn invoke a range of knowledge structures and problem solving methods to satisfy them. These include; further knowledge in production rule format, structured templates or prototypical knowledge, invocation of processes and programs such as elicitation or machine learning techniques, knowledge that allows metareasoning and analysis of the knowledge so far gleaned, or methods to allow direct call outs to user to supplied goals and solutions etc.

In the early stages of KA the approach we will exemplify relies on the use of template or stereotypical knowledge. Using these sorts of knowledge structures we aim to establish values

| | |
|---|---|
| IF | empty knowledge base |
| THEN | goal = characterise task |
| | |
| IF | goal = characterise task |
| THEN | goal = elicit verbal task description |
| AND | goal = elicit task model |
| | |
| IF | goal = elicit task model |
| THEN | goal = elicit I/O model |
| AND | goal = elicit input features |
| AND | goal = elicit output features |

Table 1: Setting up KEW goals

for a fixed set of expected characteristics of the task.

Thus static knowledge about the expected nature of information comprising any task could be used to drive something like the canned interview shown in episode 1.

**scenario** *XFRAC - Episode 1*

| | |
|---|---|
| **KEW:** | *Can you give me a description of the task the expert performs in words?* |
| **USER:** | *The expert identifies the type of fracture of a steel component in order to determine precise causes and possible remedies so as to avoid future failures* |
| **KEW:** | *Can you identify sub-tasks in this application domain?* |
| **comment** | The user would have the option of switching to a tool here if the application area was a complex array of tasks. In this case the user decides to enter sub-tasks as a list |
| **USER:** | *Analyse failure. Determine cause. Determine remedy.* |
| **KEW:** | *Could you indicate which sub-tasks are to be performed by the proposed system?* |
| **USER:** | *The task of the target system is to analyse fracture types.* |

One would now like to proceed to establish in detail what problem solving type our application *analyse fracture types* is likely to belong to; diagnosis, planning, monitoring, prediction etc. For the sophisticated knowledge engineer it might be possible to determine this by presenting them with the range of conceptual models of problem solving types known by the system (in the style of ASKE [Patel, 1989]) and asking them to select the most appropriate.

If this is not possible KEW would have to continue the questioning to try to establish for itself the problem solving type. Effectively this requires that KEW ask key questions, obtaining *metaknowledge* about the problem space. On the basis of this knowledge, it should be possible to eliminate or confirm the presence of candidate problem solving types in the application. Just such a *canned* interaction is shown in Episode 2. We should note that determining the most appropriate problem solving category for the application is itself a standard form of classification.

**scenario** *XFRAC - Episode 2*

**KEW:** *Is the goal to establish a property of the object of analysis or a classification of it?*

**USER:** *Classification.*

**KEW:** *Is the goal of the task to determine or monitor behavioural properties of the objects of analysis?*

**USER:** *No.*

**comment** KEW has now established that the application of the target system is classificatory, and would display its findings thus far to the knowledge engineer. The system now attempts to refine its own classification by trying to differentiate between analysis based on heuristics as opposed to some deeper model of the domain

**KEW:** *Is the object of analysis a device or system that can be modelled in terms of interacting components? For example, an electrical circuit, a mechanical system or subsystem such as an internal combustion engine*

**USER:** *No.*

**KEW:** *I will now try and determine the nature of the output of the system. What is the type of output of the system?*

**USER:** *Fracture types.*

**KEW:** *Estimate the number of fracture types. (An order of magnitude estimate is OK, ie 10s, 100s, 1000s.)*

**USER:** *10s.*

**comment** KEW now has the hypothesis that Heuristic Classification is a possible model for this task.

**KEW:** *Will the expert quickly generate hypotheses when confronted with a case of fracture?*

**USER:** *Yes.*

**KEW:** *Is an expert available for a KE session?*

**USER:** *Yes.*

**comment** KEW now has as its goal establishing the structure of the solution space, it chooses a laddering tool on the basis of the size of the solution set and the assumption that a hierarchy of solution types can be enumerated.

We have already indicated that aspects of KEW's activity will be model driven. Models of generic problem solving types, tasks, devices and processes might all be used to structure the acquisition process. As we shall see in the next section the selection of heuristic classification as a candidate problem solving type in the above interaction allows us to use the content of such a model to direct acquisition in a number of ways.

# 5 Using models to direct acquisition

The classic form of the heuristic classification inference structure is shown in figure 3. This shows us both the inputs and outputs of various processes that constitute the problem solving. Thus there are four types of input/output class (known as meta-classes); data, findings, solution abstractions and solutions. The crucial processes that operates on these meta-classes are; abstract, match and refine. Each of these processes is given a definition and set of methods that can be used to effect the change from input to output. Thus refine is defined as as *a process of refining solution abstractions into more specific solutions.* A method which can effect this is specialisation.

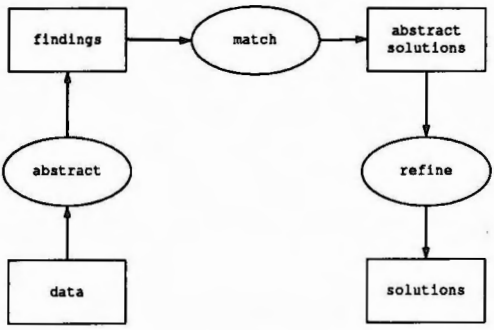



Figure 3: The heuristic classification inference structure

Just such knowledge can be matched to the appropriate tools to explore the various components of the heuristic classification model, which is taken, on the evidence to date, to be an appropriate characterisation of problem solving in this domain.

So to explore the solution space we note that specialisation is a process that can be explored using a laddering KA technique. Mapping aspects of inference models to particular KA techniques is part of the knowledge that KEW possesses.

| | |
|---|---|
| IF | current inference model = heuristic classification |
| AND | input space size = order of 10 expl |
| AND | output space size = order of 10 expl |
| AND | expert is available |
| THEN | invoke laddering and/or card sort on the solution space |
| AND | invoke laddering and/or card sort on the data space |
| AND | IF cases THEN invoke AQ11 on the matching space |

Table 2: Selecting tools

Use of a laddering tool provides an idea of the structure and interelation of solutions in the application domain. What is the size and complexity of the solution space? Is the solution space hierarchically organised? Is there a single such space?

Laddering of such a space could be immediately followed by sorting or rating of solution instances to determine the underlying attributes of the solutions. This may lead to abstractions over solutions likely to be of use if the problem solving model is indeed most appropriately characterised as heuristic classification.

# 6 Tool use in KEW

Figure 4 represents a laddering session with an expert on the solution space. The expert may or may not have added attributes to these concepts. We will assume that he did not.

KEW now has various options, which include:

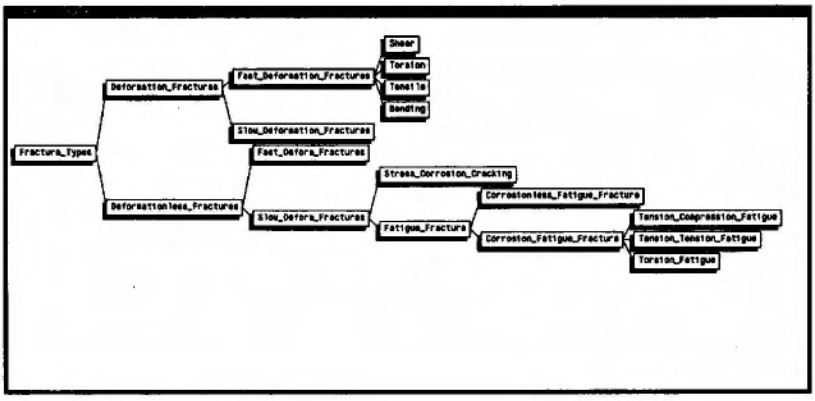



Figure 4: A laddering session with KEW

- establish the attribute structure of the nodes in the hierarchy of fracture types
- use a card sort for checking and further exploration of the fracture type space
- analyse the structure of the input space
- explore cases if they are available

On invocation of a tool we expect KEW to be active in suggesting ways in which the tool should be used. This knowledge is gleaned from the practical experience and experimental results on the use and efficacy of various forms of knowledge acquisition. Below we have the kind of knowledge that could direct and guide the use of laddering tools.

- Context - laddering the solution space
- If there is a problem in the seed item select a concrete solution/diagnosis
- First build a structure of items 10-20 - fill in attributes later
- Be clear about what the links mean - is_a, part_of, depends_on
- If a single structure doesn't fit start new laddering tool processes and build lots of local structures
- To fill in attributes - start at top and inherit down, or at the bottom with instances and ladder up

The other method which KEW suggested for exploring the solution space of heuristic classification was the card sort. The expert might generate several sorts. Figure 5 shows an example of one of the sorts generated by the expert.

KEW is now able to enter an evaluative phase in the development of the knowledge base.

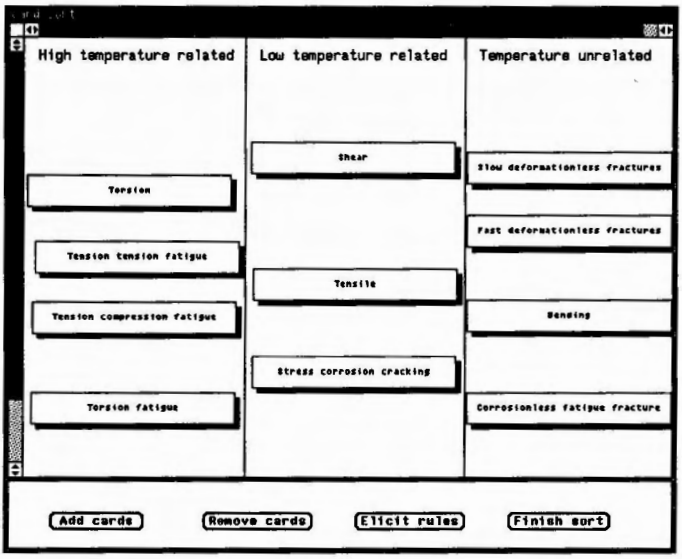



Figure 5: An example card sort with KEW

# 7 Knowledge analysis

In episode 3 we are concerned to show how KEW can act as a check on the veracity and consistency of what is elicited in a KE session. Knowledge based analysis of the knowledge elicited acts as a form of quality control in KEW.

**scenario** *XFRACT-Episode 3*

**KEW:** *Analysing your sort on* **No deformation** *versus* **Deformation.** *This sort corresponds to the two subtrees* **Deformationless_Fracture** *and* **Deformation_Fractures,** *except for* **Bending.** *Is* **Bending** *a sub-class of* **Deformation_Fractures?**

**EXPERT:** *yes*

**KEW:** *Do you agree that* **Bending** *should go in the sort pile* **Deformation?**

**EXPERT:** *OK.*

**KEW:** *I cannot relate your* **Temperature** *sort to the laddered hierarchy - is* **Temperature** *a relevant attribute for establishing* **Fracture_Types?**

**EXPERT:** *Sometimes.*

**comment** KEW could now pursue the differences in the way the concept of temperature partitions the solution types.

In episode 3 KEW analyses some the knowledge elicited so far and points out a possible error to the expert. KEW would need to have knowledge about how to relate information from one tool to that of another. This may be done by knowing how one type of knowledge maps onto another. For example the groups in a card sort are likely to map onto the attributes from the laddering session. If the expert doesn't make that mapping then KEW may pursue the issue. Table 3 shows an example rule in this situation. A further interesting feature of the second sort referred to in episode 3, and shown graphically in Figure 5, is the fact that some attributes may partition the objects of the fracture type space differently than the simple

| | |
|---|---|
| IF | sort_criterion used is not explicit in class hierarchy |
| AND | sort_criterion is not used as attributes of the classes |
| THEN | question expert as to applicability of sort_criterion |

Table 3: Knowledge analysis

sub-class structure elicited in the first laddering session. Such attributes are sometimes of sufficient importance to warrant investigation in their own right. Thus an attribute such as temperature is an example of an environmental factor. This could well form the root of a hierarchy of considerable complexity.

There are other ways in which the quality of the emerging KB may be improved by system initiated meta-level reasoning. One example present in our evolving KB is differentiating between different classes in the hierarchy. A diagnostic problem often requires that two similar objects be differentiated. KEW could examine the class hierarchy generated by the laddering tool with exactly this in mind. Control knowledge about the laddering tool tells it that it can potentially find news areas of knowledge by examining differentiating attributes. Table 4 below shows how this might be achieved.

| | |
|---|---|
| REPEAT | CHOOSE a similar pair of classes |
| | WHERE a similar pair have the same parent; |
| | IF the slot and slot value are |
| | the same for all slots on both classes |
| | THEN question expert on the differences between them |
| | FINALLY choose another pair |
| UNTIL | no similar pairs remain |
| | |
| IF | expert was questioned about differentiating similar pairs |
| THEN | ladder with the differentiating attribute as seed item |

Table 4: Differentiating between objects

# 8 Machine learning: tool use revisited

Having explored something of the nature of the fracture space KEW may now seek to understand the nature of data which is used as input in the analysis of fracture types. Again KEW is directed by the use of the heuristic classification model. In episode 5 KEW attempts to elucidate the nature of relevant data. These would be observable features and any other inputs. One option for KEW is to collect and analyse cases. In XFrac cases were available in machine readable form.

In any use of case material it would be important to establish the nature of the cases and what they exactly refer to. This might be conducted in the interactive style shown in episode 5.

**scenario** *XFRAC Episode 5*

**KEW:** *Are any cases available?*

**USER:** *Yes*

**KEW:** *Do they apply to* **Fracture_Types***?*

**USER:** *No*

**comment** This has established the important fact that cases refer primarily to objects other than fracture types.

**KEW:** *What subconcept of* **Fracture_Types** *do the cases apply to?*

**USER:** **Zone_Types**

**KEW:** *How does a* **Zone** *relate to a* **Fracture***?*

**USER:** *A* **Zone_Type** *is* **a_part_of** *a* **Fracture.**

**KEW:** *Can a* **Fracture** *have more than one* **Zone***?*

**USER:** *Yes*

**KEW:** *Can a* **Fracture** *have* **Zones** *of different types?*

**USER:** *Yes*

**KEW:** *Are* **Zone_Types** *different from* **Fracture_Types***?*

**USER:** *No*

**comment** KEW has now established a significant part of the domain model, i.e. that a fracture consists of various zones. The model for heuristic classification has to be adapted and refined to cater for this fact.

**KEW:** *Does* **Fracture_Type** *classification involve comparing different* **Zones***?*

**USER:** *Yes*

This episode also reflects the underlying goal of KEW to elaborate the task structure underlying fracture analysis. The sort of control knowledge driving this is shown in Table 5.

| | |
|---|---|
| IF | class hierarchy and attributes have been investigated |
| AND | cases are available |
| THEN | use a machine learning tool on the cases |
| | |
| IF | using a case-based machine learning tool |
| THEN | investigate which area of the domain the cases relate to |
| | |
| IF | relevant part of domain for cases is not in hierarchy |
| THEN | establish relationship between top-level object and relevant part of domain |
| AND | pursue importance of this part of domain to overall task |
| | |
| IF | relevant part of domain is important to task |
| THEN | re-model the task to include this part of it |
| AND | apply the machine learning to the cases |

Table 5: Task and domain modelling

After episode 5 KEW has to refine and verify the new model of the task: first zones will need to identified and be classified. Subsequently the fracture type needs to be established. On the basis of knowledge of various task types which are held in templates by KEW, a new model will be hypothesised. It is likely that KEW will not be able to construct a new model without help from the user. A tool for editing graphical inference structures will allow the user to modify and adapt the model suggested by KEW. It is now possible for KEW to return to its previous goal of analysing case data in order to establish what is important in a case for the purposes of zone identification.

So in episode 7 the knowledge engineer decides to look for heuristic association rules in the case data using the AQ11 machine learning method. We see that KEW has knowledge about the input requirements of various tools and in some the cases the means to transform existing data into the required format.

**scenario** *Episode-7*

**USER:** *Use AQ11*

**KEW:** *In order to use AQ we have to transform the cases into a single bundle of attribute value pairs. Can we assume that the influence of different* **Loads** *are independent?*

**USER:** *Yes.*

**KEW:** *I will transform the* **Load** *attributes into new attributes which are the combinations of* **Load_Type** *and* **Load_Variance**. *The number of cases is quite small, I suggest that you concentrate on some attributes that are suggestive of the* **Fracture_Type** *of a* **Zone**.

**USER:** *OK.*

**KEW:** *Which attributes are suggestive of* **Fracture_Type** *of a* **Zone**?

**EXPERT:** **Pattern, Roughness, Light_Reflection, Edge_Appearance.**

**KEW:** *Do you want me to automatically select candidate rules, or do you want to select them manually?*

**USER:** *Automatic*

**comment** KEW comes up with a number of rules representing heuristic associations.

**KEW:** *The following heuristic associations have been generated:*

```
ASSOCIATION_1:  ROUGHNESS = ROUGH_ABRASIVE &
                PATTERN = RAYS
              SUGGESTS FRACTURE_TYPE OF ZONE = BRITTLE

ASSOCIATION_2:  PATTERN = GRANULAR &
                LIGHT_REFLECTION = SHINY
              SUGGESTS FRACTURE_TYPE OF ZONE = BRITTLE

ASSOCIATION_3:  ROUGHNESS = ROUGH_ABRASIVE &
                PATTERN = GRANULAR
              SUGGESTS FRACTURE_TYPE OF ZONE = DUCTILE

ASSOCIATION_4:  ROUGHNESS = VERY_SMOOTH &
                PATTERN = RAYS
              SUGGESTS FRACTURE_TYPE OF ZONE = BRITTLE
```

```
ASSOCIATION_5:  ROUGHNESS = GENERALLY_ROUGH &
                PATTERN = NO_PATTERN
              SUGGESTS FRACTURE_TYPE OF ZONE = DUCTILE

ASSOCIATION_6:  ROUGHNESS = SMOOTH_IRREGULAR
              SUGGESTS FRACTURE_TYPE OF ZONE = FATIGUE

ASSOCIATION_7:  ROUGHNESS = VERY_SMOOTH &
                LIGHT_REFLECTION = DULL
              SUGGESTS FRACTURE_TYPE OF ZONE = FATIGUE
```

Although the number of cases is really too small to obtain reliable results with pure induction, the rules induced by AQ can be used as a basis for formulating the associative rules that associate data or abstracted evidence to hypotheses about the type of zone.

Another important aspect of KEW's meta-knowledge is that of transformation. When transforming knowledge between two different representations it is vital that meaning is consistently preserved and that as little is lost as possible. This issue is well illustrated in episode 7, which considers the use of the AQ machine learning technique. Knowledge that might be used by KEW in this situation is shown in table 6.

| | |
|---|---|
| IF | transforming attributes |
| THEN | ensure that they are independent first |
| AND | check the range of their influence through the class hierarchy |
| AND | check that all possible values for the attributes have been considered |
| | |
| IF | transforming relations |
| AND | relation is caused_by |
| THEN | check that all inherited preconditions have been considered |

Table 6: Knowledge transformation

# 9 Knowledge refinement

Knowledge refinement also involves meta-knowledge, one aspect is reasoning about gaps in the domain. Episode 8, is an interaction showing refinement of the results of the AQ machine learning technique. In the case of improving rules, this means that KEW has to know something about the implications of its rules. So if a rule with a particular conclusion has attributes in its conditions, these attributes have to be investigated to see whether they apply generally or just to the cases looked at. These issues are illustrated in table 7.

**scenario** *XFRAC Episode-8*

| | |
|---|---|
| IF | refining knowledge |
| THEN | try to use generalisation |
| | |
| IF | generalisation is not useful |
| THEN | try differentiation |
| | |
| IF | using generalisation on rules |
| AND | there are attributes in the rule conditions |
| THEN | find generality of each attribute value |
| AND | find if attribute always occurs |
| | |
| IF | finding generality of attribute value |
| THEN | see if the attribute value is always true for the conclusion |
| | |
| IF | finding whether attribute always occurs |
| THEN | ask if the attribute always holds in the class to which the rule applies |

Table 7: Knowledge refinement by generalisation

**KEW:** *Shall we attempt to refine our rules?*

**USER:** *Yes*

**KEW:** *We will start with generalisation. Let's look at the rules for* **Brittle Fracture_Types.** *Do* **Brittle** *fractures always have a* **Roughness** *with a value* **Rough_Abrasive?**

**EXPERT:** **Rough_Abrasive** *or* **Generally_Rough.**

**KEW:** *Is there always a* **Pattern** *in* **Brittle** *fracture types?*

**EXPERT:** *No, but often there is a* **Distinctive Pattern.**

**KEW:** *If there is a* **Pattern,** *what suggests* **Brittle** *fracture types?*

**EXPERT:** **Granular, Rock_Candy, Chevron_Marks** *or* **Rays.**

**KEW:** *I now have new associations:*

```
ASSOCIATION_10:  DEFORMATION = NOT_PRESENT &
                 ROUGHNESS = ONE_OF
                 [GENERALLY_ROUGH, ROUGH_ABRASIVE] &
                 PATTERN = ONE_OF
                 [RAYS, GRANULAR, ROCK_CANDY, CHEVRON_MARKS]
              SUGGESTS FRACTURE_TYPE OF ZONE = BRITTLE

ASSOCIATION_11:  DEFORMATION = NOT_PRESENT &
                 PATTERN = ONE_OF
                 [RAYS, GRANULAR, ROCK_CANDY, CHEVRON_MARKS] &
                 LIGHT_REFLECTION = SHINY
              SUGGESTS FRACTURE_TYPE OF ZONE = BRITTLE
```

This type of generalization could be combined with SEEK2 related techniques that test the knowledge base with respect to the cases. The converse situation to generalisation is often required in knowledge base refinement. The next episode shows an interaction where discriminative information is acquired and specialisation takes place.

**scenario** *XFRAC Episode-9*

**KEW:** *Association-10 will also apply to cases 4 and 6. I will try to refine the concept* **Brittle** *in order to discriminate between these cases and the* **Brittle** *cases. The* **Axial_Zone_Angle** *seems a candidate attribute to discriminate between* **Brittle** *and* **Ductile** *cases. Is that correct? If so, what values can* **Axial_Zone_Angle** *have for* **Brittle** *fractures?*

**USER:** **Axial_Zone_Angle = Transverse**

**KEW:** *Are there other attributes that discriminate between* **Brittle** *and* **Ductile** *fractures?*

**USER:** *Yes,* **Temperature = Low**

KEW can store this information in the form of a rule such as:

```
IF CURRENT_HYPOTHESIS_SET = [BRITTLE, DUCTILE] &
   AXIAL_ZONE_ANGLE = TRANSVERSE &
   TEMPERATURE = LOW
THEN PREFER BRITTLE
```

In this last episode we have applied similar ideas in refining rules that we applied to the results obtained from the laddering tool earlier. KEW does this by looking for and suggesting possible discriminating attributes. The sort of meta-knowledge that might perform this is shown in table 8. After a number of sessions like the ones described above, KEW

| | |
|---|---|
| IF | using differentiation |
| THEN | look for a discriminating attribute between the two conclusions |
| IF | looking for a discriminating attribute |
| THEN | check frames representing rule conclusions |
| IF | checking frames representing rule conclusions |
| AND | no discriminating attributes exist |
| THEN | ask user for attribute to discriminate |

Table 8: Knowledge refinement by discrimination

has established relations for generating hypotheses about zone types and for discriminating between zone types. These relations can be converted into a format that can be used by a generic heuristic classification reasoning module in KEW or in some target system. This would enable the knowledge engineer to test the module for zone classification. In further stages of the knowledge acquisition process KEW would support the construction of a full fracture type classification knowledge base.

# 10 Towards a conceptual model for KEW

In the previous sections we have illustrated a vision of a KBS for knowledge acquisition support. In this section we will take a more abstract view and address the question as to what the nature and structure of the knowledge in a KA support system could be. Our analysis of this knowledge will start from the global model of the knowledge acquisition process that we sketched in section 2.

A first type of knowledge needed in KEW is *planning knowledge*. This type of knowledge concerns the decomposition of the KA process and knowledge of how to solve a particular problem in the knowledge base by setting up a number of goals. Figure 6 shows an example of a decomposition of the goal *establish problem solving model*. In fact this tree represents only one way of decomposing the problem: it assumes that a predefined model can be identified from a library of models. This is not necessarily always the case. An alternative way would

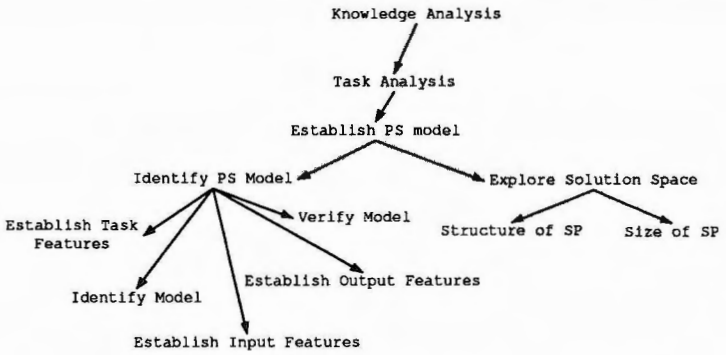



Figure 6: An example goal tree generated by the planner

be to construct a model of the problem solving process, e.g. by modifying an existing model. So, KEW must have ways to handle plans with alternative goal decompositions. Techniques used in non-linear planners such as NONLIN will be needed here.

The goal decomposition knowledge in KEW serves several purposes. First it can be used to suggest a way forward to the user of KEW. In this role the planning knowledge is the basis of advice and guidance that KEW can give. The second purpose of the decomposition knowledge is to ensure that KEW will select the appropriate operation to be performed on the application knowledge base. The leafs of the goal tree should correspond to the elementary operations that operate on the application knowledge base.

The second role of the planning process is to devise plans for remedying problems in the application knowledge base. Problem descriptions produced by the evaluation process will have a number of different elements. First they indicate the partition of the application KB in which the problem occurs. This may be a single suspect knowledge element, a pair of knowledge elments that are in conflict or need to be merged, or it may be a large set of knowledge elements. A second element of the problem description is the type of knowledge involved, e.g. concept definition, rule, fact, isa-hierarchy etc. Identification of this type is necessary in order for the planning knowledge to differentiate between different remedial plans for different types of knowledge. For example, inconsistent vocabulary can usually only be resolved through interaction with the knowledge engineer, whereas there are other

means of handling inconsistent concept hierarchies. The third type of information in a problem description concerns the type of problem. The choice of a remedial plan will often be difficult. Techniques employed in systems such as SEEK2, BLIP, TEIRESIAS and INDE+ might be used here. In many cases the planner will have to consult the user.

Once a goal to be achieved has been established by the planning component, one or more operations have to be selected by the *selection processes.* In many planning approaches this process is viewed as a subprocess of the planning itself. In our conceptual model of KEW selection of operations is separate since it may involve the user. The selection process is driven by knowledge about the operations that achieve a certain goal and about preconditions and external factors that may influence the choice. For example, the goal to establish a concept hierarchy can be best achieved using a laddering technique, but when no expert is available and written material describing the structure of domain concepts is at hand, a tool like the Shelley's concept editor may be a better choice.

The selection process is viewed in KEW as a heuristic classification task. The data about the current state of the acquisition process are abstracted and together with the goal produced by the planner used to generate hypotheses about potential operations. The operations are structured in a hierarchy and a refinement/verification process selects the most appropriate operation. The major classes of operators are:

- Knowledge Modelling

  construct the 'meta'-knowledge about the domain knowledge and the required reasoning process:

  - selection from library
  - description space construction
  - interactive modification
  - knowledge differentiation
  - feature elicitation

- Knowledge Creation

  create new knowledge element

  - direct elicitation (asking for knowledge)
  - indirect elicitation (use special techniques)
  - interactive analysis
  - inductive inference

- Knowledge Modification

  modify existing elements in the repository

  - dwimify
  - example based learning
  - transformation
  - integration
  - interactive modification
  - merging
  - deletion

- heuristic pruning

Note that the operations that KEW knows about are not necessarily complete knowledge acquisition techniques. The approach that we propose here is to separate out elementary operations from techniques were possible in order to keep as much explicit control over the knowledge acquisition process as possible. For example an induction technique such as AQ11 [Mitchell *et al.*, 1986] can be split into several more elementary operations: hypothesis generation, hypothesis ordering and hypothesis selection. By separating these operations we will be able to configure the optimal sequence of operators in the light of knowledge about the application domain that has already been gathered.

Once a set of knowledge elements has been created or modified by an operation it has to be assimilated into KEW's information base. This requires three types of processes: indexing, type identification and linking. The indexing is necessary to keep a knowledge dictionary uptodate. Type identification is input for the evaluation process and possible operations which will need the new knowledge as input. Links between knowledge elements are kept to allow KEW and its user to track down the acquisition history of knowledge elements.

The final process that we have to consider is *evaluation.* There are many ways in which the knowledge base as a whole and individual knowledge elements can be evaluated. The following list gives an overview of some of the possibilities [vanSomeren, 1989]:

- correctness with respect to a set of test cases
- completeness with respect to a set of test cases
- completeness with respect to conceptual model
- internal consistency
  - vocabulary
  - logical consistency
  - isolation
  - structural consistency (inheritance)
  - consistency with respect to test cases (no contradictory solutions)
  - consistency with respect to conceptual model
  - consistency with knowledge templates (cf. Teiresias)
- redundancy
- adequacy with respect to the target user
- adequacy with respect to the expert or KE (e.g. comprehensibility, acceptability, explanation)

A detailed investigation of the different ways of evaluating knowledge is one of the current activities in the ACKnowledge project.

Admittedly, the conceptualisation sketched above is still rather high level. On the other hand, the scenario and the examples of the different types of knowledge that KEW employs show how the details of the conceptual model can be filled in. The construction of a number of prototypes is currently underway to test and further develop the framework described here.

# 11 Conclusion

In this paper our view of how we will implement KEW has been described. It is based on a framework that views knowledge acquisition as a series of activities whose aim is to construct a sequence of models and associated information structures.

KEW is envisaged to be an *active* and *directive* support tool for the knowledge engineer. This will be effected by construing KEW as a KBS for KA. Its knowledge sources will consist of; knowledge of the KA process itself, knowledge about the capabilities and limitations of various KA methods and techniques, knowledge of of generic models, and knowledge about general processes of evaluating, refining and integrating the emerging partial knowledge bases that will ultimately constitute the application KBS.

The analysis of possible interactions between tools, as exemplified in the scenario, has led to a model of the operation of KEW which is an iterative process of planning KA. These plans consist of cycles of activity; selecting a particular tool to invoke, constructing a partial KB, evaluating the new knowledge base, refining the KB on the bases of the evaluation, and entering a new cycle of tool selection.

The construction of the scenario has been a revealing exercise. It has shown the enormity of the search space in knowledge acquisition. At every moment in the KBS development process the knowledge engineer has many options available. Even for experienced knowledge engineers it is not obvious what the best route through the KA space is. KEW's purpose will be to attempt to steer the knowledge engineer through the space.

It has become clear that major elements in KEW will include the ability to use meta-level knowledge. This meta-level knowledge will be about KA in general, tools in particular, and the analysis of emerging IKBs. KEW will engage in active inference on aspects of the knowledge engineering activity and its emerging results.

We should reiterate the point that various components of the KEW described in this paper have been implemented. Of course, the complete vision presented here is a description of our ambitions rather than of a fully implemented system. We believe that our analysis of KEW is viable, and that the functionality inherent in the scenarios can be realised. We are actively engaged in building prototypes that comprehensively embody the ideas of this paper.

# Acknowledgements

We are grateful to members of the ACKnowledge team for the help in developing the ideas in this paper. In particular, we would like to thank Maarten van Someren, Cleo Julien, Han Reichgelt, Geir Aakvik, Einar Dehli, Francisco Garijo, Robert de Hoog, Nigel Major, Jean Charles Marty, Ole Jacob Mengshoel, Pedro Luis Munoz, Wilfried Post, Emilio Rodriguez, Luis Solorzano.

# References

[Anjewierden, 1987] A. Anjewierden. Knowledge Acquisition Tools. *AI*

*Communications*, 0:29–38, 1987.

[Bennet, 1983] J. S. Bennet. Roget: A knowledge-based consultant for acquiring the conceptual structure of an expert system. Stanford Heuristic Programming Project Memo HPP-83-24, Standford University, 1983.

[Boose, 1985] J.H. Boose. A knowledge acquisition program for expert systems based on personal construct psychology. *Int. J. Man-Machine Studies*, 23:495–525, 1985.

[Boose, 1986] J. Boose. Rapid acquisition and combination of knowledge from multiple experts in the same domain. *Future Computing Systems*, 1:191–216, 1986.

[Breuker & Wielinga, 1989] 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*, pages 265–296, Amsterdam, 1989. North Holland.

[Burton *et al.*, 1987] A.M. Burton, N.R. Shadbolt, A.P. Hedgecock, and G.Rugg. A formal evaluation of knowledge elicitation techniques for expert systems: Domain 1. In D.S. Moralee, editor, *Research and development in expert Systems IV*, pages 136–145, 1987.

[Burton *et al.*, 1988] A.M. Burton, N.R. Shadbolt, G. Rugg, and A.P. Hedgecock. Knowledge elicitation techniques in classification domains. In *Proceedings Of The 8Th European Conference On Artificial Intelligence*, pages 85–90, Munich, Germany, August 1-5 1988.

[Bylander & Chandrasekaran, 1988] T. Bylander and B. Chandrasekaran. Generic tasks in knowledge-based reasoning: The 'right' level of abstraction for knowledge acquisition. In B. Gaines and J.Boose, editors, *Knowledge Acquisition for Knowledge Based Systems*, volume 1, pages 65–77. Academic Press, London, 1988.

[Chandrasekaran, 1987] B. Chandrasekaran. Towards a functional architecture for intelligence based on generic information processing tasks. In *Proceedings of the 10th IJCAI*, pages 1183–1192, Milano, 1987.

[Clancey, 1985] W.J. Clancey. Heuristic classification. *Artificial Intelligence*, 27:289–350, 1985.

[DeMarco, 1982] T. DeMarco. *Controlling Software Projects.* Yourdon Press, New York, 1982.

[Eshelman, 1989] L. Eshelman. MOLE: A knowledge-acquisition tool for cover-and-differentiate systems. In S. Marcus, editor, *Automating Knowledge Acquisition for Expert Systems*,

pages 37–80. Kluwer Academic Publishers, The Netherlands, 1989.

[Friedland, 1979] P. E. Friedland. Knowledge-based experiment design in molecular genetics. Report STAN-CS-79-771, Stanford University, 1979.

[Klinker *et al.*, 1987] G. Klinker, C. Boyd, S. Genetet, and J. McDermott. A knack for knowledge acquisition. In *Proceedings of the 6th National Conference on Artificial Intelligence*, pages 488–493. Morgan Kaufmann, July 13-17 1987.

[Marcus *et al.*, 1985] S. Marcus, J. McDermott, and T. Wang. Knowledge acquisition for constructive systems. In *IJCAI-85*, Los Angeles, 1985. Morgan Kaufmann.

[Mitchell, 1983] T.M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, 1983.

[Mitchell *et al.*, 1986] T.M. Mitchell, R. Keller, and S. Kedar-Cabelli. Explanation based generalization: a unifying view. *Machine Learning*, 1:47–80, 1986.

[Morik, 1987] K. Morik. Acquiring domain models. *International Journal of Man-Machine Studies*, 26, 1987.

[Morik, 1989] K. Morik. Sloppy modelling. In K. Morik, editor, *Knowledge Representation and Organisation in Machine Learning*. Springer Verlag, 1989.

[Motta *et al.*, 1989] E. Motta, T. Rajan, and M. Eisenstadt. A Methodology and Tool for Knowledge Acquisition in Keats-2. In P. Guida and G. Tasso, editors, *Topics in the Design of Expert Systems*, pages 265–296, Amsterdam, 1989. North Holland.

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

[Nelson, 1980] T. Nelson. Replacing the printed word: a complete literary system. In *Information Processeing 80*, pages 1013–1023. Amsterdam:North Holland, 1980.

[Newell, 1982] A. Newell. The knowledge level. *Artificial Intelligence*, 1982:87–127, 1982.

[Patel, 1989] J.M. Patel. Aske. Technical report, Open University, 1989.

[Shadbolt & Burton, 1989] N.R. Shadbolt and A.M. Burton. Empirical studies in knowledge elicitation. In *SIGART Special Issue on Knowledge Acquisition.* ACM, 1989.

[Shortliffe *et al.*, 1981] E.H. Shortliffe, A.C. Scott, M.B. Bischoff, A.B. Campbell, W. van Melle, and C.D. Jacobs. Oncocin: An expert system for concology protocol management. In *AJCAI-81*, pages 876–881, 1981.

[vanSomeren, 1989] M. van Someren. A classification of techniques for knowledge acquisition. Esprit Project 2576 Deliverable ACK-UvA-T1.2-DL-004A/C, University of Amsterdam, 1989.

# Cases, models or compiled knowledge? – a comparative analysis and proposed integration

Maarten W. van Someren, Li Ling Zheng and Wilfried Post

Department of Social Science Informatics (SWI)
University of Amsterdam
Herengracht 196
1016 BS Amsterdam
The Netherlands
E-mail: maarten@swi.psy.uva.nl

**Abstract**

*This paper describes three paradigms for knowledge acquisition: the* case based paradigm, *the* explicit model paradigm *and the* compiled knowledge paradigm. *The differences are described in terms of available information sources, required functionality and knowledge representation scheme. The differences are illustrated by three acquisition processes on the same domain. The results are used to show how an architecture for an integrated system that uses various types of knowledge can be designed together with a corresponding knowledge acquisition plan.*

## 1 Introduction

The design of first generation expert systems was motivated primarily by the goal of effective problem solving. These systems are intended to encode the knowledge of an expert who can directly, without much thought, recognize the solution to a problem. Almost on second thought, explanation facilities are added. Prospective users of the system often do not trust or even understand the conclusion produced by the system and want to be able to check the arguments for a conclusion. The explanation facility uses the solution trace as a basis. It is presented to the user in a dialogue structured around *how* and *why* questions. The representation languages most frequently used to construct such systems are production rules (often with certainty factors) and frame-based languages. The knowledge is acquired by asking the expert for the conditions under which a particular conclusion was valid. The system is constructed incrementally, using a (growing) set of test problems as a criterion and as a trigger for further elicitation. Production rules are an adequate representation language,

*We want to thank Michael Sramek for his expertise, Irma Volkers for help with the case-based reasoning system, Ameen Abu-Hanna and Richard Benjamins for their comments on a draft of this paper and the participants in the ACKnowledge project, in particular Bob Wielinga, for many discussions which inspired this paper.

because they are close to the way in which experts express justifications and conclusions. (See [5] for an extensive discussion of the history of the MYCIN system, which pioneered and exemplified this paradigm.) However, in many cases, this approach based on direct elicitation of rules, simple rule application (eg. by backward chaining) and explanation directly from the reasoning trace, turns out to be inadequate (eg. [7], [9], [26]). Because the system was designed to solve problems, much knowledge that was not directly relevant for finding the solution was left out completely or appeared implicitly in the way in which knowledge was formulated in the knowledge base. This applies both to knowledge about the domain (details of causal processes, unusual arguments, etc.) and to knowledge about control of the reasoning process. In addition to that, engineering the system (debugging, maintenance) of large, but poorly structured compiled knowledge can become a major problem. We shall call the paradigm sketched here the **compiled knowledge paradigm**, because its focus is on knowledge that is effective in problem solving and that is often available in a (more or less) "compiled" form in human experts.

The drawbacks of this paradigm have led expert system builders to incorporate explicit models of the domain and of the reasoning process (eg. [8], [24]). We shall call this the **explicit model paradigm**. A KBS (Knowledge-Based System) containing explicit models can produce much more detailed and comprehensive explanations. In addition to that, explicit models allow a decomposition of the knowledge base that is more difficult to achieve in compiled knowledge systems. This is very important in engineering the system (eg. [4]). Although explicit models can be represented in production rules, existing systems are often written in special representation languages (eg. [10], [14], [3]). We shall refer to such systems as *deep models*, for lack of a better term and contrast them with *compiled models*, which contain only knowledge that is optimized for performance in problem solving. Deep models for diagnosing technical faults or recognizing disease processes are much more expensive to run than *compiled* knowledge bases (eg.[3], [24]) and knowledge acquisition and the actual construction of the system can be very difficult. It will often involve the acquisition and representation of much more varied and structured knowledge than in the *compiled* paradigm.

Recently, a third approach has appeared as a practical possibility. This approach relies primarily on cases: problems for which an expert solution is available. We shall refer to it as the **case-based paradigm**. There are two forms. In the first, called *case-based reasoning* (eg. [20]), the knowledge base consists of a set of past case records and problem solving consists basically of matching the old cases with a new problem and using a notion of similarity to produce a solution. Domain knowledge may be available for use in the matching process, as in PROTOS or CASEY ([21]). Problem solving may of course involve a considerable amount of search, if the set of cases is large. In the second form *induction* techniques are used to construct a knowledge base from the cases (eg. [19]). This is what is commonly known as "machine learning" (a bad term, because machine learning involves more than induction). We shall use the term *case-based paradigm* to refer to both case-based reasoning and induction.

Within each of these paradigms, there is of course a wide range of techniques, but we still believe that there are characteristic differences between them. These involve a choice for

- the *function* of the knowledge (eg. problem solving and/or explanation)
- the type of *information source* and knowledge elicitation (eg. asking for conditions under which a conclusion is valid, formalizing domain theories, models and problem solving strategies and collecting cases)

- the class of *knowledge representation schemes* (e.g. production rules, special modelling languages, cases plus some analogy-based inference technique)

In other words, each paradigm commits the knowledge engineer to a function and a type of information source, with associated elicitation techniques and representation schemes.

In practice, however, a knowledge engineering task may not fit well into any of these paradigms. Eg. a complete textbook and a description of the domain may be available, which suggests the *explicit model paradigm*, but no serious explanation may be required from the KBS. Here the construction of an explicit model may well result in overkill. Another possibility is, that both a set of cases and a human expert with excellent compiled expertise are available. This suggests two paradigms: the *compiled knowledge paradigm* and the *case-based paradigm*. The knowledge engineer now faces a choice between two paradigms, which may result in the option of not using either the domain expert or the cases. Finally, it is well possible that each type of information source is available, but only for part of the domain or for part of the entire range of problem solving tasks. Eg. in medical diagnosis domain, there may be an expert with both compiled and explicit knowledge about a subclass of the relevant diseases, some cases of patients which overlap in part with the diseases of the expert, for some other diseases, the underlying processes may be unknown, even to experts, etc.. This again raises a problem. We argue here, that

1. each of the "pure" paradigms is like a package deal that is well suited for a substantial class of knowledge engineering problems, but that
2. a cost-benefit analysis of application of each paradigm should precede the knowledge acquisition process. This analysis should include the required functionality (including maintenance and refinement) and the available information sources and it should propose an architecture for the KBS and a global plan for the knowledge acquisition (integrating the paradigms).
3. knowledge acquisition methods should be developed that make it possible to use different types of information sources
4. architectures should be designed that make it possible to incorporate different types of knowledge (ie. knowledge obtained from different types of sources) and that support different functions (eg. problem solving and explanation)

In this paper, we report on experience with each of these paradigms on a single domain. This gives a first indication of the costs and benefits of applying each of these paradigms in a pure form. Next we describe our current proposal for an architecture that integrates each of these knowledge types.

To obtain a better idea of the trade-offs involved in each of these approaches, problem solving and knowledge acquisition, we constructed three versions of a knowledge base, using each of the "pure" approaches. It is of course not possible to fully control such an experiment and we therefore we only make a global comparison, along the following dimensions:

- What is the level of problem solving performance?
- What kind of explanation can be obtained from the knowledge base?
- How do the time and space complexities compare?

- How easy is it to acquire the knowledge?
- How easy will it be to perform future maintenance and refinement of the knowledge?

We first describe the domain and the functions and information sources that are relevant here. Next we present the experience with the three paradigms and finally we show how the strengths and weaknesses can be used for designing a knowledge acquisition plan plus an architecture for a KBS.

# 2 The domain

The problem for which a KBS is to be constructed here, is early diagnosis of acute chest pain on the basis of data obtained by a telephone interview. The most frequent diagnoses are angina perctoris, myocardial infarction, arrhythmia, pulmonary edema and functional complaints (probably related to anxiety). The domain contains considerable uncertainty and even experts find it hard to give a diagnosis on the basis of the data that are available here. Once a patient is taken to hospital, extra measures are taken (including cardiogram, blood tests and general physical examination), but our problem concerns the situation before a patient is taken to hospital.

# 3 Information sources

There are of course many different types of information sources that can be used in knowledge acquisition. In the context of our three paradigms, the important points are:

- *Explicit models vs. compiled knowledge* Here we may further distinguish between explicit models of inference processes and descriptive models of the domain. By compiled knowledge we mean that no (or very few) intermediate results are inferred between givens of a problem and its solution. We believe that the dimension *explicit - compiled* is independent of other characteristics such as qualitative vs. quantitative, causal vs. empirical, common sense vs. expertise, deep vs. shallow, etc.. Both explicit models and compiled knowledge may have each of these characteristics.
- *General knowledge vs. specific experiences* Note that this dimension is independent of the previous one: one may have a very detailed and explicit representation of a single problem with its solution, including the arguments and the solution process, or, on the other hand, a general "compiled" rule that directly relates a diagnosis to data about a patient.
- *Form* The knowledge may be in a standardized, structured form or in general text. Another important point is of course, if it is directly available or if it has to be elicited from the expert.

There are several sources of information in the chest pain domain:

**Documents** These include medical textbooks and handbooks about cardiology and related diseases. These contain general explicit model knowledge in textual form.

**Human experts** Beside medical specialists (cardiologists), psychologists have some expertise about functional complaints and eg. the personnel of the ambulance service who answer the telephone have some expertise. These experts have both compiled and explicit knowledge, both general and specific knowledge and they can express some of it in natural language.

**Case records** Descriptions of patients who were inteviewed using a standard questionnaire which was designed early during knowledge acquisition. Questions include complaints, disease history, etc. This knowledge is of course relatively specific, compiled and available as structured data.

# 4 Representation schemes for diagnostic reasoning

There are of course many possible representation schemes for representing diagnostic knowledge. The best-known candidates for this problem are: production rules (particularly for compiled knowledge, but also for cases and perhaps for explicit models), causal networks or qualitative reasoning models (for explicit models). Representations of explicit reasoning strategies are a metalevel architecture (eg. [23], [15] [2]) and problem solving models (eg. [6], [12]). For our discussion, the representation language is not a central issue, although it may have an important effect on the ease with which an actual KBS can be constructed. However, we believe that each representation scheme can be applied within each of the paradigms.

The knowledge that needs to be represented in the chest pain domain consists of *descriptions of cases* (in each paradigm), *compiled relations between case descriptions and solutions*, *a causal model of various disease processes in terms of symptoms, disease states and processes, causal factors.* There are several models of diagnostic reasoning processes which can be used to represent the reasoning strategy (eg. [8], [4], [6]).

# 5 Functions of KBS

In this context we only need a global classification of functions. Two types of functions that do not need comments are *problem solving* and *explanation* (including justification, trace of the reasoning process). Our third function, *knowledge acquisition*, needs some introduction. It may be the case that further refinement or extension of the knowledge base is necessary. In many cases, this is done using the existing knowledge base. Eg. case records can be used to debug and refine the knowledge base.

All three functions are potentially relevant in the context of diagnosis of chest pain. Problem solving consists of finding the best diagnosis for a given case and explanation refers to a more detailed presentation of the relation between the case and the solution. The nature of the explanations can of course vary in granularity and in subareas. Eg. the details of variation in pressure and flow rate of the blood that are caused by the narrowing of the arteries need not be explained here. The level required of explanation depends on the intended user. Here we take the level of students of medicine or nurses as the target and explain processes in the style of undergraduate medical textbooks.

Further knowledge acquisition is a requirement here, because it must be possible to adapt the knowledge base to the kind of clients involved. The current prototype is developed in

the context of an emergency heart care unit at a hospital, but the final version needs to be used by a general ambulance service. It is well known that there are substantial differences between countries, medical schools, hospitals and doctors in how they deal with such patients. In addition to that, there is a considerable research effort which may produce new relevant knowledge. For these reasons it is important to be able to modify the knowledge base.

Note that new cases will be relatively easily available, since they can be collected during application of the system, or during an initial stage where only the interface of the system is used, but not the knowledge base. It is possible that new explicit knowledge about the domain becomes available as a result of scientific research.

# 6 Three paradigms applied

The three paradigms were all applied to the chest pain domain. One important problem, which information about a patient is relevant for this problem, was solved separately by interviewing experts and studying the literature in this domain. This resulted in some 35 questions to be asked, most of which had a range of possible answers. These questions were used to collect the cases, to elicit compiled knowledge and they also played an important role in the construction of the causal model.

## 6.1 The case-based paradigm

The cases are descriptions of patients with their associated diagnoses. Some 350 case records were collected from patients who were referred to the hospital emergency heart care unit. The interviews were, therefore, taken after the onset of the complaint. The hospital gave us access to the final diagnosis for the patients. The diagnoses refer to the moment that the patient left the emergency heart care unit, which is at most 24 hours after entering it. The diagnoses of the ischemic diseases are probably very reliable, because they are based on an extensive physical examination. The diagnosis "functional complaint" may be less reliable.

### Content

The description of a case contained only the data that were available from the patient by telephone plus the final diagnosis. It did not contain other information about pathology, etc.. The description looks like this:

**diagnosis:** inferior infarction
**sex:** male
**age:** 61 - 70
**manifestations:** pain – duration (1 hour - 4 hours)
location (midchest, left arm)
onset (at rest)
type (sharp, constricting)
severity (severe)
**historical fact:** experienced pain before, smoking,
hypertension

**disease history:** diabetes, previous infarct
etc.

### Representation and Reasoning

There are several ways to use cases directly in a KBS. The first is to use *case-based reasoning* and the second is to apply some induction technique to infer rules from the cases and then use the rules. We followed both paths. A simple case-based reasoning technique was applied and also an inductive learning technique. (In fact several induction techniques were used, see [18] for a comparative discussion).

There are several versions of the case-based reasoning paradigm. We used a "best-match" method: the answer to the most similar case in the knowledge base is taken as the answer to the problem . Matching used simply the number of corresponding features.

Retrieving similar cases can be expensive in terms of time. Time complexity depends directly on the number of cases, the size of a case description and the matching function. It can be improved by an appropriate indexing scheme (eg. a decision tree or even a hashing scheme) but we shall not discuss the implementation details here. Also the space taken up by the knowledge base depends directly on the number and size of the case descriptions.

The *inductive learning* version of this paradigm was done using implementations of AQ, ID3 and logistic regression.

### Problem solving

Performance was evaluated by a series of cross-validation experiments. Performance of each of these techniques was around 65 percent correct on cross-validation. (See [18] for a comparison of the induction techniques and a more detailed report on the experiments.)

### Explanation

In case-based reasoning an explanation is provided by retrieving the case on which the answer is based, referring to the matching algorithm and possibly by showing that the new case is different from other cases. This is clearly inadequate. More explanation could be provided by presenting some more data about the patient, such as the pathology and disease history, if that were available. The explanation on the basis of pure case descriptions or on the rules produced by the induction techniques is worse than that from cases, because the rules produced by the techniques are difficult to understand. This is because, unlike the cases the rules do not correspond directly to meaningful knowledge elements of the user.

### Knowledge Acquisition

The existing cases, used by the computer program in problem solving, were collected by students who interviewed patients at the hospital. The actual interview takes about 3 minutes and a whole session can be done within 5 to 10 minutes. Patients arrived at an average rate of about 25 a week at the hospital. Collecting all cases took about a year of very simple and low intensity work.

Applying the induction techniques involved some experiments with parameters of the techniques, but this can be done in a few days.

### Maintenance

No further maintenance has taken place, but it is obvious that maintenance using new cases would not cause problems, unless the number of cases would become very large. It is easy to add new cases and run the induction programs again. However, refining the rules on the basis of other knowledge is hard.

### Strength and Weakness

If there is enough supply of cases (or these are readily available), then collecting cases is a relatively cheap way of obtaining information about a domain. In this domain it took several weeks to find a good set of descriptive terms (effort that is required in each paradigm). The major weakness of this approach seems to be poor explanations and poor maintenance from other information sources than cases.

## 6.2 The explicit modelling paradigm

By explicit modelling we mean modelling of fundamental domain knowledge. The explicit model of a domain may include different types of fundamental domain knowledge such as, in a physical domain, the function, the structure, and the behaviours (state transitions) of devices. Our model only contains pathophysiologic knowledge, that is knowledge about changes in structure and function of the body that are related to a disease.

### Content

Knowledge is represented as pathophysiologic states which are the models of abnormal internal body conditions. This modelling was done to a certain depth, including more detail was felt to be useless for diagnosis.

For each state the causes, effects, manifestations and corresponding diagnosis are given. For example, state *shortness of blood in heart muscle* is represented as below:

**manifestation:**
symptom: pain – location (midchest, left chest, ..),
type (choking, pressuring, ...),
onset (at work, at rest, ...).
finding: enzymes rise in blood, ..
**cause state:** obstruction of coronary artery
**effect state:** ischemia, ...
**diagnosis:** myocardial infarction
etc.

The relationship between any two adjacent states is a causal relation (in a broad sense). Linking all states together through the causality will construct pathophysiologic processes

of diseases. In this representation, manifestations are associated with states. Some states correspond to diseases and a subset of these have the status of *diagnosis*. Theses can be the answer of a diagnostic problem.

### Representation and reasoning

States in knowledge base were represented as frames. The inference mechanism consists of constructing a paths consisting of state chains, which explains the observations as manifestations of a series of states.

The relations in this knowledge structure are only plausible relations, eg. *shortness of blood supply* may cause *heart muscle injuried* which may in turn manifest *pain in midchest*. Thus inference provides us with a set of plausible diagnoses. We could not use an existing shell for this and therefore the implementation is not finished yet. As a result we can only make tentative claims about performance and explanations.

### Explanation

The knowledge structure is able to explain a diagnosis, as a causal path, presenting the various intermediate states and their relations. But for some observation, such as *smoking* which is viewed as risk factor in the model, the reasoning process has to pass through a long way from initial states such as *heperin disappear* to the *heart muscle dies* in order to explain the relation between *smoking* and *myocardial infarction*.

### Acquisition

The explicit model, used by the problem solver, was acquired from a textbook and a domain expert. After reaching the first version of domain knowledge, we posed a series of question in lights of the acquired knowledge from the book and showed these questionnaires to the domain expert before interviews. The final version of knowledge is the sum of the literature knowledge and expert's knowledge. The process took about two months of elicitation and analysis.

A major problem in the acquisition was to obtain measures for probabilistic relations between pathophysiologic states, manifestations and states. These relations (e.g between states) are, to some extent, restricted by temporal relations which are difficult to evaluate. In fact, this problem has not been fully solved yet.

### Maintenance

Although we have no new experience (after building the knowledge base) with maintenance, it seems rather easy to add new states, that appear to be relevant, new manifestations, etc., if these are presented in explicit form. This has happened several times in the later stages of acquisition and gave no problems.

### Strengths and Weaknesses

The strength of this paradigm is clearly in explanation and in maintenance from new explicit domain knowledge. The main weakness is, that, at least without applicable tools, it is difficult formalize the knowledge.

## 6.3 The compiled knowledge paradigm

The compiled knowledge is optimized with respect to a particular task. All pieces of knowledge that are not strictly needed for problem solving are left out and others are transformed to make them more efficient. Using it, human experts are often able to immediately recognise the solution to a problem, without explicit reasoning about the underlying processes. In our case a domain expert (a junior doctor) defined a simple set of rules for recognizing the various diagnostic categories. These were validated on the cases that were available. Using feedback about how many cases were correctly classified by the rules and how many cases couldn't be classified at all, the expert tried to improve the rule set.

### Content

The expert tried to formulate his knowledge in short rules, initially covering the rules of thumb about the domain. Later he refined the rules against the cases. No intermediate categories were used, nor causal relations, etc. The following are some of the rules that were found.

```
URGENT:
  ((complaints = pain) V (pain = yes))
  & severity_pain = severe
  & dyspnoe = no
  & familiar_with_pain = yes
  & similarity_pain = (same V worse)
  & nitroglycerine_effect = no_effect
OR
  ((pain_location = right_arm) V (pain_irradiation = right_arm))
  & severity_pain = severe

NOT_URGENT:
  ((complaints = pain) V (pain = yes))
  & dyspnoe = no
  & familiar_with_pain = no
  & severity_pain = (moderate V low)
  & pain_location = left_chest
OR
  ((complaints = pain) V (pain = yes))
  & familiar_with_pain = no
  & complaints = palpitations
```

### Representation and reasoning

The rules were represented as simple production rules in terms of the case descriptions. No attempt was made to structure the knowledge base or to introduce intermediate knowledge. After a number of experiments, a set of about 10 rules was designed. Neither modifying the individual rules nor adding new rules to this set did raise performance on the cases that were available.

### Problem solving

Performance of the rules was around 87 percent correct for the cases that can be classified by them, which was 50 percent of the test cases. If in the undecided cases the class with the greatest a priori probability is taken, performance on the complete case set goes is 67 percent correct. Applying the rules is quite fast.

### Explanation

Explanation based on the rules is not impressive but exceeds the level of the case-based paradigm. This is probably because the expert is focussed at each rule on a recognizable and managable facet of the domain, which therefore is at a more appropriate level of explanation. The rules provide a somewhat better basis for explanation than those in the case-based paradigm.

### Acquisition

The rule base was constructed incrementally by a domain expert, a junior doctor. He wrote down rules, inspired by his knowledge about typical cases and the existence of some rules of thumb. The format for the case descriptions was the same as in the other paradigms. He implemented the rules and tested them on the cases. The results, in terms of how many cases were correctly classified and how many remained undecided, were used to refine the rules. A part of this acquisition proces was to find a balance between a rule set that was maximal complete, i.e. that could classify as many as possible cases, and a rule set that was maximal correct. After about a week no significant progress was made and acquisition was stopped. It can be continued with new cases, or new diagnostic categories. However, it has now become very difficult to keep track of the effect of changes and the expert felt that not much progress was possible.

### Strength and Weakness

Problem solving performance here is about the same as in the case-based paradigm. Explanation based on the rules exceeds the level of the case-based paradigm. The time needed to produce the system is similar to that needed for case-based methods.

# 7 Comparison of the three kinds of knowledge

On the basis of our experience with the three kinds of knowledge, we can make a comparison of their strengths and weaknesses. In general this confirms current ideas about these paradigms. Different knowledge engineers were involved and the performance levels of the knowledge bases were not completely controlled. Also the reader should keep in mind that we used "pure" versions of the three approaches and only a single example. Eg. using a different modelling language for the deep model would have produced different results and there are ways of indexing and annotating cases that affect the speed of the reasoning process and the quality of explanations. Also, no sophisticated tools were used to support acquisition. We believe, though, that using the pure forms of these techniques brings out their characteristic differences. In the next section we discuss the possibility of an integration of these paradigms.

## 7.1 Problem solving

The performance level obtained by the three approaches was approximately the same. Response times for cases and compiled knowledge are similar and very short. The prototype of the model is not yet fully operational, but it will definitely be considerably slower.

An important point is the way in which models deal with uncertainty. As we saw, experts find it very difficult to provide probabilities for use in the deep model and the effects of changes tend to be difficult to track down. It is also difficult to find an adequate modelling language for uncertain deep models. Compiled knowledge has been generalized from cases. It tends to be less sensitive to small changes.

## 7.2 Explanations

If the original descriptions of existing cases contain detailed information, case-based reasoning is capable of providing rich explanation about a new case. Otherwise, the explanation contains only "shallow" information. Because making decision bases on the assessments of similarity, the quality of a conclusion is restricted by how to evaluate the similarity. Conversely, deep knowledge reasoning guarantees the quality of a conclusion and provides rich explanation.

Although we have not formally evaluated this, explanations from cases seem to be better than those from compiled rules, because the basis of the rules is often unclear.

## 7.3 Acquisition

In this domain, acquisition of cases is rather easy. Therefore, the case-based paradigm and the compiled paradigm are relatively cheap. However, in the explicit modelling paradigm, the cases were not used. If we compare the (approximate) time taken by each paradigm, the case-based paradigm was probably the fastest and the other two were similar. [1]

[1] The delay in completing the deep model was mainly due to the problems with implementing a representation scheme for modelling uncertainty.

## 7.4 Maintenance

We are interested in maintenance of the knowledge, rather than the implementation. There are two sources of new knowledge: new cases (in particular cases on which an error was made) and new explicit model knowledge, resulting from scientific research. New cases are most easily accommodated in the case-based paradigm and to a lesser extent in the compiled knowledge paradigm, because at some point it becomes rather difficult to keep track of the effect of changes to the knowledge base. There is no good method for refining explicit models from cases (although some attempts along this line are being made (eg. [11], [27]).

Incorporating new explicit domain knowledge is relatively easy in the explicit modelling paradigm, but in both other domains it is clearly difficult.

## 7.5 Summary

| | *Case-based* | *Explicit model* | *Compiled* |
|---|---|---|---|
| *Performance:* | 65-72 | ?? | 67 |
| *Explanation:* | Poor | Good | Poor |
| *Size:* | Large | Large | Small |
| *Speed:* | Good | Bad | Good |
| *Elicitation effort:* | Low | High | Low |
| *Modelling effort:* | Low | Very high | Low |
| *Case-based refinement:* | Easy | Hard | Feasible |
| *Adding new theory:* | Hard | Easy | Hard |

# 8 Integrating the three paradigms

We argued that the three paradigms are each most suited to a particular combination of required functionality, available information sources and representation scheme. In practice one may encounter situations where none of the paradigms is quite appropriate. The chest pain domain is an example. Consider the problem with accommodating new cases and new explicit domain knowledge. None of the pure paradigms can do this. We believe that in that case a mixture of paradigms should be used and that an architecture for the KBS should be designed *together* with a knowledge acquisition plan. This should detail for each of the components in the KBS how the knowledge for it will be acquired. The mixture should be such that each component, each sub-KBS, fits one paradigm in terms of required functionality and available information sources. In addition to that, we need an architecture for combining the knowledge involved in applying each of these paradigms, in a uniform representation scheme. There are several proposals for systems that integrate "deep" and "compiled" knowledge in one representation. Eg. Sticklen and Chandrasekaran ([25]) distinguish varying levels of detail and describe a representation scheme that can represent and reason with multiple levels. However, these architectures are motivated by a trade-off between efficiency and explanation and the architecture supports both fast reasoning (at abstract levels) and the possibility to produce detailed reasoning for explanation or for dealing with problems that are not covered by the abstractions. Our focus is on knowledge acquisition and we need the combination to be able to incorporate knowledge that is available

either in compiled form (eg. in a human expert), in the form of case descriptions or as an explicit model (eg. as in a manual or a textbook).

The structure that we propose is one that seems effective for at least some medical diagnosis domains, is an architecture that consists of a sequence of filters. Each filter is able to reduce the set of possible diagnoses or at least to obtain a more refined estimate of their probabilities.

Each filter is a sub-KBS, to be constructed by one paradigm, which suits its requirements and available information sources. How the filters are defined (and how many) is an engineering problem, for which we have no guidelines, except that they should be homogeneous in the relevant properties.

We illustrate this approach with the design for an advice system for acute chest pain. The design that is presented here has not been implemented yet. The idea in this domain is to divide the knowledge into the following components:

**Prototypical cases** These are patients that can directly be diagnosed, using generally accepted knowledge that is shared by experts and taught to students. Diagnoses that are based on this knowledge need no explanation. This fits very well with the *compiled knowledge paradigm* and also with the *case-based paradigm*, though less clear.

**Unusual but clear cases** These are patients with an unusual description, that cannot be directly diagnosed as a standard case. In general some diagnoses can be excluded, but there several candidate diagnoses, none of which is clearly the best. However, by using knowledge about the pathology (and possibly about terminology, measurements, and maybe about other circumstances) they can still be assigned a clear diagnosis. The diagnosis will need explanation in terms of pathology, circumstances, terminology, etc.. Acquisition of this knowledge will follow the explicit modelling paradigm.

**Unclear cases** The remaining cases can be characterized as the difficult ones that cannot be clearly diagnosed. Here statistical arguments from epidemiological studies come into play. A limited form of explanation may be useful, but in terms of the arguments pro and contra a diagnosis. Knowledge acquisition will rely on cases, following the *case-based paradigm.*

A similar analysis was given by David *et al.* [13], but emphasizing explanation rather than knowledge acquisition. He summarized some second generation expert systems into 5 types including using compiled knowledge to face routine problems, generate original results. Explicit models are used to solve atypical problems, test the original results and give deeper explanation. In the other words, compiled knowledge heuristically suggests original hypotheses and models act as backup or justification.

# 9 Discussion

The conclusions from our analysis, which is consistent with the experience in knowledge acquisition in the chest pain domain, are that:

- There are three paradigms for building KBS that each have their strengths and weaknesses.

- It is useful to make a distinction between the *knowledge* that is represented and the *representation scheme*. The acquisition process and the functions of the knowledge do not depend directly on the representation scheme.
- An analysis of the required functionality and of the available information sources should precede the actual knowledge acquisition. It can be used to design an architecture for the KBS and to plan the corresponding knowledge acquisition process.

The distinction between knowledge types is consistent with psychological descriptions of human experts. Eg. according to Boshuizen et. al. ([1], [22]), subjects at different levels of medical expertise apply functionally different knowledge structures in solving a case. Novices depend more on explicit so called *basic-science* knowledge that explains signs and symptoms, resident students depend more on *pathophysiologic* knowledge. And experienced physicians depend more on *compiled* knowledge, they also often mention specific cases from their own experience or from the literature as knowledge that they use to solve a diagnostic problem.

An interesting problem, raised by our view of knowledge acquisition, is if and how the acquisition process can be supported by techniques for machine learning. We briefly discussed one application: induction techniques for constructing compiled rules from cases. However, there are several other interesting tasks. An important task seems to be refinement of explicit knowledge bases from cases. Systems that can be a basis for this are eg. INDE ([27]), DISCIPLE ([16]) and BLIP ([17]).

We found that **uncertainty** is an important problem in diagnostic problem solving. Both reasoning with uncertainty and the acquisition of uncertain knowledge are problematic. We even believe that this may be so important that it should become a distinctive feature of a paradigm. In particular, since some statistical data (from epidemiological studies) are available about prevalence and about factors predicting some of our diagnostic categories. For the *explicit modelling paradigm* uncertainty raises substantial problems, because it is difficult to elicit uncertain information from experts, because uncertain deep models tend to be very expensive computationally and because they are very hard to engineer by hand. There is some anecdotal evidence that eliciting uncertain knowledge frome experts is unwieldy and should be avoided. Both the *case-based paradigm* and the *compiled knowledge paradigm* seem to have fewer problems with uncertain knowledge.

# References

[1] H.P.A. Boshuizen, H.G. Schmidt, and D. Coughlin. On the application of medical basic-science knowledge in clinical reasoning: implications for structural knowledge differences between experts and novices. *the 10th Aunual Conference of the Cognitive Science Society,17-19 Aug., 1988, Canada*, 1988.

[2] B. Bredeweg. Introducing meta-levels to qualitative reasoning. *Applied Artificial Intelligence*, 3-2:85–100, 1989.

[3] B. Bredeweg and B.J. Wielinga. Integrating qualitative reasoning approaches. In *Proceedings of ECAI-88, Munich*, pages 195–201, 1988.

[4] 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*, pages 265–296, Amsterdam, 1989. North Holland.

[5] B.G. Buchanan and E.H. Shortliffe. *Rulebased Expert Systems: The Mycin Experiments of the Stanford Heuristic Programming Project.* Addison Wesley, Reading, Massachusetts, 1984.

[6] B. Chandrasekaran. Towards a functional architecture for intelligence based on generic information processing tasks. In *Proceedings of the 10th IJCAI*, pages 1183–1192, Milano, 1987.

[7] W.J. Clancey. The epistemology of a rule based system -a framework for explanation. *Artificial Intelligence*, 20:215–251, 1983.

[8] W.J. Clancey. Heuristic classification. *Artificial Intelligence*, 27:289–350, 1985.

[9] W.J. Clancey and R. Letsinger. NEOMYCIN: Reconfiguring a rulebased expert system for application to teaching. In W.J. Clancey and E.H. Shortliffe, editors, *Readings in Medical Artificial Intelligence: the first decade*, pages 361–381. Addison-Wesley, Reading, 1984.

[10] J.H. de Kleer and J.S. Brown. Theories of causal ordering. *Artificial Intelligence*, 29:33–61, 1986.

[11] G. DeJong. Explanation-based generalization with plausible inferencing. In K. Morik, editor, *Proceedings EWSL-89*, pages 1–10, London, 1989. Pitman.

[12] L. Eshelman. MOLE: A knowledge-acquisition tool for cover-and-differentiate systems. In S. Marcus, editor, *Automating Knowledge Acquisition for Expert Systems*, pages 37–80. Kluwer Academic Publishers, The Netherlands, 1989.

[13] J. M. David et al. Foreword. *Ninth International Workshop, Avignon, France*, 1989.

[14] K.D. Forbus. Qualitative process theory. *Artificial Intelligence*, 24:85–168, 1984.

[15] P. Jackson, H. Reichgelt, and F. van Harmelen. *Logic-Based Knowledge Representation.* The MIT Press, Cambridge, MA, 1989.

[16] Y. Kodratoff and G. Tecuci. Techniques of design and DISCIPLE learning apprentice. *International Journal of Expert Systems*, 1:39–66., 1987.

[17] K. Morik. Sloppy modelling. In K. Morik, editor, *Knowledge Representation and Organisation in Machine Learning.* Springer Verlag, 1989.

[18] W. Post and M.W. van Someren. Een vergelijkende evaluatie van enkele technieken voor inductief leren. In L. Kwee at.al., editor, *Proceedings AI Toepassingen*, Amsterdam, 1988. NGI.

[19] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[20] B. Porter R. Bareiss and C. Wier. Protos - an exemplar based learning apprentice. *Int. J. Man-Machine Studies*, 29:549–561, 1988.

[21] S.P. Ross. Case-based reasoning: position paper. *Proceedings of the SCAI 1989*, 1989.

[22] H.G. Schmidt, H.P.A. Boshuizen, and Pie P.M. Hobus. Transitory stages in the development of medical expertise: The "intermediate effect" in clinical case studies. *the 10th Aunual Conference of the Cognitive Science Society,17-19 Aug., 1988, Canada*, 1988.

[23] B. Silver, editor. *Meta-Level Inference.* Elseviers Science Publishing B.V., Amsterdam, 1986.

[24] L. Steels. Second generation expert systems. *FGCS*, 1(4):213–221, 1985.

[25] J. Sticklen and B. Chandrasekaran. Integrating classification-based compiled level reasoning with function-based deep level reasoning. *Applied Artificial Intelligence*, 3:275–304, 1989.

[26] W.R. Swartout. XPLAIN: A system for creating and explaining expert consulting systems. *Artificial Intelligence*, 20:285–325, 1983.

[27] P.P. Terpstra and M.W. van Someren. Inde: a system for knowledge refinement and machine learning. In Boose J., Gaines B., and Linster M., editors, *EKAW'88*, pages 30–1 30–8, Bonn, 1988. Gesellschaft fuer Mathematik und Datenverarbeitung MBH.

# *First Order Logic Foundation of the KADS Conceptual Model*

Thomas Wetter
IBM Germany, Scientific Center
Institute for Knowledge Based Systems
Wilckensstr. 1a, D-6900 Heidelberg, FRG
EARN: WETTER at DHDIBM1

## Abstract

KADS is the most influential approach to knowledge engineering in Europe in the last few years. KADS regards knowledge engineering as a modelling activity and clearly distinguishes phases, each producing a special kind of model. The result of the first phase in KADS is the conceptual model. It serves as mandatory interface between the expertise related acquisition activities and the artefact related design activities. For reasons of clarity and in order to function as a common tool for persons specialized in acquisition or in design, the conceptual model should have a formal notation in a semantically founded language which has sufficient expressiveness to cover the constructs from all layers of the KADS conceptual model. This paper presents a semantic foundation for such a language. It starts with 1st order predicate logic (**1stPL**) with syntactic variants for concept heterarchies and procedure attachment as a base language. Subsequently, elements of the KADS conceptual model are represented in this formalism. A precursor version of the formalism has been used in a project on balance analysis. For exemplary tasks in this domain it could be shown that system design could be carried out on the basis of this formal notation by persons without specialized AI skills.

## 1 Introduction

After six years of ESPRIT funded research, KADS ([25]) has evolved into the most influential approach to knowledge engineering in Europe ([8; 23; 15; 10; 9]). Its influence can also be found in other systematic approaches to knowledge engineering such as [16; 12] Some of KADS' distinguishing features are that it aims at being domain independent (in contrast to domain related workbenches such as OPAL [13] or KNACK [11]) and independent of knowledge representation formalisms (in contrast to e.g. PRED [27]).

### *1.1 Phase concept and models*

A main theme of KADS is that knowledge acquisition is a modelling activity. Models are incorporated in a phase concept which aims at conducting knowledge "handling" in a way similar to conventional software engineering. Different models serve as deliverables between distinct phases. Ideally, this allows for job and skill sharing in such a way that persons are specialized to conduct specific phases, i.e. can read the input model of a phase and can produce its output model but need not know about any other activity or the re-

sult of any other phase of the whole process of knowledge acquisition. Concretely, KADS speaks of a:

- **conceptual model** as a knowledge level model of the domain expertise
- **abstract design model** as a functional description of the artefact to be built
- **technical design model** as a detailed description in terms of design elements of the artefact to be built

For the purpose of this text, we do not discuss the additional **phases** of implementation etc. and the additional **models** describing the EDP environment for the future artefact and the user system cooperation (although none of the following is in conflict with these additional requirements), but concentrate on the conceptual model.

### *1.2 Purpose of the conceptual model*

In KADS, the conceptual model is organized in four layers:

| | |
|---|---|
| **domain** | application specific static knowledge |
| **inference** | elementary application independent methods |
| **task** | major subproblems to be solved in the domain |
| **strategy** | control of solution process |

From the perspective of knowledge acquisition as the first major phase, the conceptual model is the final document which is produced in a number of iterations and which fixes every identified item that was judged to be relevant for covering the contents of the application or the expertise to be modelled. From the perspective of system design i.e. the second major phase, the conceptual model is the mandatory input concerning the involved knowledge. In other words, the conceptual model is ***the*** interface between content related acquisition activities and artefact related design activities. This establishes a central role for the conceptual model if we take the perspective again that knowledge acquisition and system design should be done by different persons. In this case, the first should be responsible for any content related decision (and exempt from any design related decision), whereas the latter should be exempt from any content related decision. This, however, means that only the entire contents and nothing more must be fixed ***unambiguously*** in the conceptual model.[1] Otherwise the system designer would, maybe unconsciously, make decisions for which he is neither skilled nor informed nor can be made responsible.

The praxis in most existing environments of KADS application is that knowledge acquisition and system design are done by the same person(s). Here, the above necessity of unambiguous formulation does not yet show up, since the system designer "understands" the conceptual model and can close gaps or dissolve ambiguities.

In the special case of the StatCons experiment ([3]) it so happened that knowledge acquisition and system design had to be done by different people. They were, however, from the core of the KADS group, i.e. knew quite well how to interpret informal KADS notations and how others would formulate the knowledge. In this special case, system design could be done by a person other than the one who had done the knowledge acquisition [21].

To be sure in the general case (i.e. with two [groups of] persons being exclusively responsible for either knowledge acquisition or system design) that the authentic contents of

---

[1] This may include deliberate decisions before or during knowledge acquisition to leave out certain details that are either judged irrelevant or resist modelling.

knowledge are communicated by the conceptual model requires either very optimistic assumptions about some universal language ([4]) or definite provisions to avoid ambiguities and resulting (mis)interpretation of the conceptual model by the system designer.

The suggested means of avoiding ambiguities and hence of preventing (mis)interpretation is a formal language for the conceptual model. This necessity of a formal notation for the conceptual model has also been acknowledged in [26] for reasons of precision, verifiability, and ease of translation. Hence, [26] have presented a formal language for the three bottom layers of the KADS conceptual layers, which they base on an intended-interpretation semantics that is, however, not specified in any more detail.

Beyond the arguments of [26] there seem to be some further reasons to plead for a formal language for the conceptual model:

- To establish quality standards already for the results of the knowledge acquisition phase, and not only for the later artefact.
- To allow for skill specialization towards expertise with respect to content related activities on one hand and towards artefact or design related activities on the other hand
- To use the formal notation of the conceptual model to specify an executable version, which will tend to be totally inefficient on one hand but on the other hand. represents a true image of the mandatory manifestation of the expertise; this can be understood as a prototype in the sense of not being an efficient production system, but as functionally isomorphic to the final system behavior.

### *1.3 Résumé of requirements*

This now allows to sum up the desiderata for the notation of the conceptual model:

1. It requires a formal language notation in order to avoid that the system designer be plagued with content related decisions.

2. The expressiveness of the language must be high in order to offer constructs for all elements of the KADS layers and their connections.

3. The semantics of the language must be well defined as a formal indication of the non-ambiguity and meaningfulness of the expressions used and in order to allow for formal checks and implementation of interpreters to result in a kind of prototypes.

These requirements are inherent.

4. To make the language useful for knowledge level description of expertise, it needs comprehensive syntactic constructs in order to be principally comprehensible for the knowledge engineer and (in a less rigorous sense) for the domain expert.

5. The language needs tool support in order to allow factual control of real application size conceptual models.

These latter requirements offer some degree of freedom.

### *1.4 Constraints and limitations*

Since we will be dealing with a high level knowledge representation language, it makes sense for the clarity of presentation to describe the language in terms of a somewhat lower, but well understood knowledge representation language. Thus one constraint arises from what **cannot** be expressed in the terms of this lower language.

The other limitation arises from what **need not** be expressed: since there is no list or description of language constructs of the strategy layer of KADS other than "top goal", "exit" (cf. [26]), there is no need at present to offer formal language constructs. This may indicate either that strategy will remain a realm of creative intelligence or that in the more or less near future we shall also understand the strategy layer in terms of some semi-formal notation. In the latter case, the attempt will have to be made to include the strategy layer in the proposed formalism.

### *1.5 Scope of this paper*

In this paper, a foundation for a formal language is suggested for the representation of the KADS conceptual model. It claims to satisfy the above requirements 1. through 3., i.e. to have a well defined semantics and to cover all constructs used in the semi-formal notations of the three lower KADS levels and the connection between them.

There will be some indication of what is meant by comprehensive syntactic constructs, but these examples should not be taken as a formal introduction of the syntax. To describe tools definitely goes beyond the scope of this paper, i.e. requirement 4. is slightly touched and requirement 5. is reserved for subsequent publications.

## 2 Prelude

### *2.1 Conceptual modelling.*

As in conceptual modelling, we have to deal with what to model and how to model. In this case, "what" deals with the informal descriptions of elements of the KADS conceptual model. "How" involves the elements of the underlying (or ***base***) representation language.

As in conceptual modelling, we have linguistic data in the form of KADS papers. We have to determine the meaning of concepts such as "concept" found in the linguistic data by means of some interpretation model. As interpretation model we use knowledge about the semantics of expressions in the base representation language, i.e. we interpret the linguistic data in terms of constructs available in the base language. Finally, we have to come up with the conceptual model, which in this case is the formal language for conceptual models. [2]

### *2.2 Outline of presentation.*

Since interpretation models are a kind of basso continuo of conceptual analysis, we start with our interpretation model, i.e. the syntax and semantics of the base language (Chap. 3). To be more illustrative, there is an intermezzo (Sec. 3.x) about a concrete banking application, which is re-used in later sections. Subsequently, concepts from the three layers will be mapped onto the constructs of the base language layer by layer, starting with the domain layer (Chap. 4). Finally we'll describe the connection between the layers in terms of the base language (Chap. 5). During the two latter steps, we'll suggest two variants for a number of constructs in order to emphasize different aspects that can be meant by the linguistic data. Finally, there is a description of the status (Chap. 6) and a discussion (Chap. 7).

---

[2] If this reminds the reader of [7]), he might decide to call the whole enterprise a fugue of which the prelude is a prelude is a prelude is a prelude is a prelude is a prelude is a

## 3 Basso Continuo - Semantic Foundation

For a number of reasons we base the whole analysis on 1st order predicate logic (*1stPL*). Among the major reasons are

- that the "meaning" of 1stPL expressions is well understood in a number of ways such as existence of models, decidability of the consequence relation including factual availability of theorem provers
- that "en vogue" knowledge representation languages such as the KL-ONE derivates can be understood as extensions of 1stPL (and have been better understood once they had been looked upon as such cf. [17])
- that, if prototyping in the above sense of an executable conceptual model becomes important at some point in time, PROLOG lends itself as a programming language which offers a number of basic functions closely related to 1stPL proof search.

### *3.1 1st Order Predicate Logic (1stPL)*

The starting point is absolutely standard 1stPL, i.e. the base language contains symbols for

- variables V,W,...
- constants c,d...
- predicates p,q,...
- functions f,g,...
- quantifiers $\forall$, $\exists$
- connectors &, |, $\neg$

For convenience of presentation, we additionally include symbols for well formed connections from these elements as an element of the language, i.e. we use

- prime and composite formulae $\pi$, $\phi$, $\psi$ ...

which are equivalent to the resp. connection. $\leftarrow$ and $\rightarrow$ are classically defined by means of | and $\neg$.

We make the usual conventions about sorts and arities of predicates and function symbols, i.e. we introduce a standard signature for the language [3]. This allows a structure, onto which the signature can be mapped, to be used to define the semantics of the pure 1stPL core of the base language. In other words: the anchoring for truth of a predicate is that its corresponding relation holds in the algebra. Completeness and correctness theorems are known for different deductive methods (resolution, sequent calculus, ...), i.e. the core language is well understood in a number of senses.

---

[3] For a systematic description of the logical foundation of the more or less informal arguments presented in the sequel, the reader is referred to a textbook of Computer Science Logic, such as [1; 5].

### *3.2 Variants of 1stPL*

Now there are two more or less obvious ways in which to vary 1stPL

- include what has been called the T-BOX in the KL-ONE family of knowledge representation schemes which can be dealt with by order-sorted logic ([17]).
- allow procedure calls from within a logic environment and proof calls from within procedures ([18]).

**Order sorted logic and concept heterarchies**
A very general view allows multiple sort relations, i.e. one sort may be sub- and supersort for different other sorts. Such relations can be expressed explicitly and implicitly by their sets of attributes. There can be atomic sorts, roles of sorts with respect to other sorts etc. To put it more theoretically: in the order sorted logic variant we have a non-trivial signature, i.e. type of variables resp. constants further determines wellformedness of expressions (besides the above mentioned arity of functions and predicates.)

### *3.x Intermezzo - A Glance at a Banking Application*

Preliminary versions of the material presented in this text have been applied in a banking project on customer balance analysis. For illustration of the essential required constructs we use one aspect of this application. The **customer** is described by formal characteristics (e.g. **legal_form**) and a number of **indicators** of its performance. Some **indices** i.e. particularly important indicators are judged by performing one or more **comparisons**. A comparison involves a target to be compared (which is the value recorded in the year to be analyzed), one reference for each comparison, with which the target has to be compared; these may be former years, the average of the branch, or some objective standard. The documentation of the **source** data is part of the **indicator** sort whereas their use is part of the **comparison** sort.

Judging an index includes computing a difference between target and reference, comparing, whether the difference exceeds a threshold, and noting whether an essential increase is directed, i.e. positive, or, negative, or whether it is neutral. For the task of customer balance analysis in a banking environment we illustrate the major domain layer constructs of the language using the concepts **customer** and **comparison**. domain layer extract.

```
sort customer
    features
        name : string
        ...
        legal_form_ : legal_form
    roles
        indicator_ : indicator
        ...
```

**customer** denotes a sort, i.e. subsequently we can introduce variables of sort **customer**. "Features" denote 1-place-properties of the sort, "roles" 2-place properties. They can be used explicitly by expressions such as legal_form(C) or indicator_(C,I), where C is of sort customer and I is of sort indicator. This demonstrates that what is used in a feature or role expression must also be specified as a sort or must be one of the standard sorts such as string.

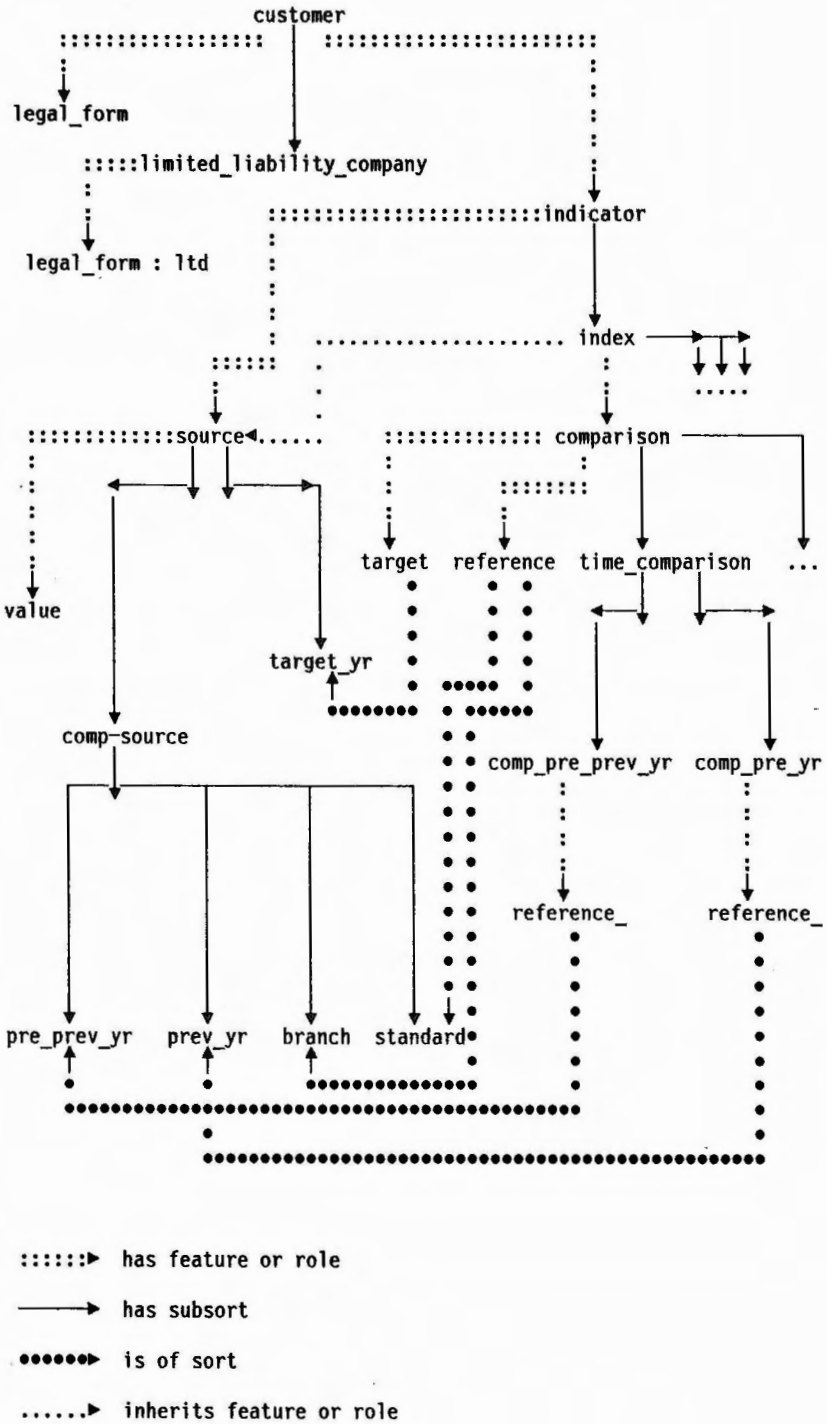



Figure 1: Graphical illustration of the following

Subsort notation may read

```
sort limited_liability_company < customer
    features
        legal_form : ltd
```

which corresponds to some

```
sort legal_form
    atoms ltd, plc, ...
```

or

```
sort index < indicator
    roles
        comparison_ : comparison*
```

The first needs a foundation on (in this case atomic) sorts for legal forms, which is provided by the second expression. The last assumes that a sort **indicator** with some technical detail such as range, unit, **source** of data etc. as features is or will be defined (see below). It introduces **index** as a special form of indicator with a set of comparisons that have to be performed for this indicator (the asterisk denotes plural or set_of.) The sort descriptions concerning all types of comparison have to express the entity or target to be judged, the reference to which it has to be compared, a sensitivity threshold, directedness, i.e. whether an increase is positive, negative, or neutral, ...

```
sort comparison
    features
        threshold : numeric
        directedness_ : directedness
        judgement_ : judgement
    roles
        target : target_yr
        reference_ : reference
    atoms
        comp_prev_yr, comp_pre_prev_yr, comp_branch, comp_standard
```

Subsorts of comparison can be described as synthesis from atomic sorts by expressions such as

```
sort time_comparison
    = { comp_prev_yr,comp_pre_prev_yr }

sort comp_prev_yr
    roles
        reference : prev_yr
```

Sources of indicator values form another part of the concept heterarchy.

```
sort indicator
    features
        ...
        source_ : source

sort source
    features
        ...
        value : numeric

sort target_yr < source

sort reference
    features
        ...
    atoms prev_yr, pre_prev_yr, branch, standard

sort comp_source < source
    atoms prev_yr, pre_prev_yr, branch
```

Finally we introduce the sort **judgement**, which is the result of a comparison or may be simply directional or evaluatory.

```
sort direction < judgement
    atoms increase, decrease, constant

sort evaluation < judgement
    atoms improve, degrade, constant
```

We can now express

- that there is a sort **comparison** and subsorts with special properties
- that any comparison is characterized by a target to be judged which has a numerical value in a certain year
- that it is to be compared to some earlier year, the average of the branch or some standard value which, as sorts, also inherit a numeric value feature from sort **source**
- that there is a threshold for the relevance of the absolute difference
- that the direction of the difference may or may not play a role
- that the result of the comparison may be simply directional or an indication of good or bad performance.

The remaining variety of supported sort expressions including available standard sort, and algebraic properties of the sort heterarchy can be found in [17]. An individual comparison to be performed by a compute-like knowledge source requires comprehension of some of these sorts w.r.t. variables of these sorts in predicates

```
comparison_form(I,C,J,X,Y,Z)
numeric_comparison(X,Y)
numeric_difference(Z)
```

which, in themselves, are not part of the sort lattice. The first predicate says that for an index I, comparison C based on values X an Y is appropriate. The relations introduced by these predicates are defined by an axiom

```
∀ I : index
∀ C : comparison_
∀ Tr: numeric
∀ T : target
∀ R : reference_
∀ J : judgement
    comparison_(I,C)
  & target(C,T)
  & reference_(C,R)
  & threshold(C)=Tr
  & value(T)=X
  & value(R)=Y
∃ Z : numeric
→ comparison_form(I,C,J,X,Y,Z)
  & numeric_comparison(X,Y)
  & numeric_difference(Z)
```

### *3.2 Variants of 1stPL (continued)*

The semantics of order sorted logic are very similar to the one of pure 1stPL, only the algebra is more complex. Concerning proof theory, the critical aspect is finding a most general unifier for terms, about which some more or less complex sort relation is known. One positive result of [19; 24] is that if the sorts form a meet semilattice, we can have complete and correct deduction methods again. The negative upper bound is outlined by [14].

Practically, the formal use of sorts entails role restriction, which has been characterized as essential for future efficiency issues in [26], but has been approached in their paper by different methods.

**Procedural attachment**

While the above variant supports straight forward notation of taxonomies or concept hierarchies, we now come to a number of seemingly non-1st-order expressions such as known, unknown and language constructs for controlling the process of evaluation and substitution. We extend the syntax by attached_procedure predicates of the form **a_p(in:X,Y, ...; out:U,V).** [4] , which can be placed anywhere in a "knowledge base" or "logical theory". They correspond to procedures which are to be called and executed with respect to the present status of a proof. Additionally, a logical proof can be called from within a procedure and parametrized by the intermediate values of variables of the procedure.

The semantics of this procedural attachment has been described in [18] as inspecting and manipulating the model for a logical theory. This approach seems to be related to the dynamic algebras of [6], i.e. can in a more abstract sense be given a proof theory. The denotational semantics of the attached-procedure predicates has to assume that there is an at most finite number of fixed distinct expression $a_p_i(\ldots)$, $1 \leq i \leq n$ (for ease of presentation the index i is subsequently omitted) and that for each of them there is a 1-1 corresponding executable procedure, which calculates the values of U,V, ... from the values of X,Y, ... . I.e. each a_p is defined by the extension of its argument-value vectors.

---

[4] a_p stands for attached_procedure predicate

In the simplest case of a 1-dimensional output U it can be understood as the graph of the function realized by the procedure, i.e. can be defined in a logic without equality. This indicates that one aspect of an attached-procedure predicate a_p is the existence of a unique predicate p whose interpretation is the graph of the function or its generalisation towards more than one-dimensional output. As a consequence, we can distinguish the following interesting cases:

1. In case of instantiated X,Y, ..., U,V, ..., i.e with constants x,y, ... u,v, ...

   **a_p(in:x,y, ...; out:u,v, ...)**

   is equivalent to the predicate

   **p(x,y, ... u,v, ...)**

   which may be true or false.

2. In case of instantiated X,Y, ..., i.e. constants x,y, ... , and free U,V, ... the unique values u,v, ... are calculated and instantiated according to the definition of the attached procedure. This automatically entails that then

   **a_p(in:x,y, ..., out:u,v, ... )**

   must be true .

3. In case of non_instantiated X,Y, ... the procedure of

   **a_p(in:X,Y, ...; out:U,V, ... )**

   is not evaluated, which is in accordance with an interpretation of p, which does not contain any tuples with free X,Y, ... .

As to the use of these variants: 2. can be seen as an easy means to make use of any sources of knowledge which are not known as declarations but in the form of algorithms, i.e to integrate procedural control in a natural way. 3. allows to deal with known-ness or unknown-ness of variables without have the need to introduce extra-1stPL expressions such as known(X).

All three cases can occur as subgoals anywhere in a superordinate logical proof. For case 2. this implies that in the course of the superordinate proof the whole graph of the function may have occured. But with any fixed substitution of X, Y, ... the substitutions for the U,V, ... are uniquely determined.

We conclude that variants of 1stPL are being used which offer a number of comprehensive constructs for complex concepts, but that we don't go beyond the edge of tractability, (at least, if we respect [19]). In other words: so far we satisfy requirement 3. and in part requirements 1. and 4. Now we come to requirement 2.

# 4 Theme - Founding the KADS Layers on the Base Language

## *4.1 Domain layer*

**Concepts** can be interpreted as sorts. This can be substantiated by taking formulations from [20] and interpret as follows

| | |
|---|---|
| **entity** | sort or instance of a sort (depending on context of using **entity**) |
| **attribute** | feature of a sort or role between sorts |
| **value** | ground substitution of a variable |
| **subsume** | sort - supersort relation [5] |

The semantics for the sort related variant of 1stPL introduced above includes that these are all available.

**Relations** are naturally mapped on predicates. This reverses that the semantics of predicates is relations, or, in other words, to express relations, 1stPL naturally offers predicates.

**Complex structures** , if not covered by sort heterarchy, will be written as formulae. Prima facie one will make the assumption that formulae in the conceptual model are true, i.e. axioms, which span a logical theory. But this assumption is not crucial, since on the higher level of the conceptual model one might also deal with explicitly arguing about truth condition as in a truth maintenance system.

Examples for these domain layer constructs can be found in the previous intermezzo Sec. 3.x.

It should, however, be stated that no type of modal logic is supported so far. If complex domains to be modelled (e.g. including time expressions or causality based inferences) require this extension of the base language for the domain layer, the whole semantic foundation will have to be re-examined.

### *4.2 Inference layer*

**Knowledge sources and metaclasses** can be viewed as functions and their (formal) arguments ([26]). In a number of cases it is equally plausible to regard a knowledge source as a relation between two metaclasses. The former view can imply directionality: in case of non-injective functions, the inverse of the function is no longer a function. Hence one might argue in favor of the latter non-directional view: if two metaclasses are in some relation, their permutation is in the inverse relation. *assemble(set,structure)* from ([20]) is a typical example of this kind. While the directional interpretation is that some set is assembled into some structure, the non-directional interpretation is that the structure is an assembly of the set, with the equivalent formulation that the set is the decomposition of the structure. Whatever the choice is, there is one alternative from the repertoire of the base language which lends itself for modelling knowledge sources:

- attached-procedure predicates
- predicates

The latter support an overall declarative style of representation, whereas the former offer additional constructs from outside the realm of pure 1stPL.

Metaclasses as formal arguments may appear too weak to cover the variety of uses of domain layer knowledge for conducting inferences. We'll come back to this when discussing the connection between layers.

---

[5] Subsume may also have a more general meaning such as in law, where complex events etc. satisfy the conditions established in some legal code. This technically means that some event is an instance of a code condition.

Some elementary inferences such as computation or comparison can hence have two distinct formalisations:

1. procedural

```
a_compute_difference
    (in:  Pos_argument, Neg_argument;
     out: Difference                  )

a_compare
    (in:  Argument, Threshold;
     out: Relevance_of_argument)
```

2. declarative

```
computation(Pos_argument,Neg_argument,Difference)
comparison(Argument,Threshold,Relevance_of_argument)
```

(**comparison** here is different from the sort, which can have two arguments as a maximum.) The difference, when discussed on the compute example is that the procedural formulation is a real computation in the case of known Pos_argument and Neg_argument and unknown Difference, a logical test when all three are known, and a fail, when one or two of Pos_argument or Neg_argument is unknown. The declarative formulation entails a proof search with backtrackable substitution of argument values, which produces similar solutions in the first two cases, but also supplies a solution with Difference known and one of the other unknown and an infinite set of solutions with more than one parameter unknown, no matter whether this relates to the arguments or the Difference.

Hence the procedural formulation allows controlling that certain inferences only be conducted with known argument values, or - from a different perspective - to use attached-procedure predicates for testing whether certain values are known. On the other hand, the normal predicates allow the formulation of the connection between concepts without giving any regard to the way in which this connection can be used.

### *4.3 Task layer*

According to [20] the task layer contains goal statements and control statements.

**Goal statements** quite neatly match the goal or head statements p in axioms of the form **p ← q & r &....** So we decide to model goals as predicates. Their predicate names indicate the kinds of goals (judge, explain ... ), whereas argument names indicate the kinds of results (judgement, reason). Finally the results will have to be related to domain entities. Elementary goals will have to be related to knowledge sources. This will be dealt with in the paragraph about connecting the layers. At present we state

1. goal statements can be formulated inside 1stPL w.r.t. its order sorted variant inasmuch as predicates or sort expressions are used

2. conjunction (**q & r & ...**) is the simplest form of control, which will subsequently be enhanced by more complex control expressions. In case of expressions such as **p ← q & r & ..., q,r,** etc. will be called subgoals of **p.**

3. it is useful for a number of reasons (and has by this time also been appreciated by the KADS group, cf comparison of [20] and earlier publications with [26]) to include the

arguments in goal expressions, e.g. to write *abstract(Feature_set,Data_set)* instead of *abstract(Feature_set)*.

**Control statements** We distinguish between two major classes

1. **if-then-else**
2. **while**

(cf. [2]) **while** may mean **as long as an index set has not been exhausted** or **as long as some other condition is satisfied**. The second aspect can be treated as a simple conditional similar to **if** .

In the scope of the conditional (if or while) can be a 1stPL or order sorted logic expression (prime formula or sort expression) or something like **known(X), unknown(X)**. The first can be dealt with by normal deduction (i.e. testing whether the predicate or sort expression is a consequence of the theory). The second makes use of the failure of an attached-procedure predicate, which takes X as an input argument.

if-then-else as a whole may also have a declarative and a procedural meaning:

***logic consequence relation*** if and then part are coincidently true
or
***production rule*** when you find the if part true, make the then part true

The former needs predicates, the latter attached-procedure predicates. The if-else combination and the "while" for conditions other than index set can be treated correspondingly.

*while(index-set)*

is somewhat different and makes the impression at first that we have to go into 2nd order predicate logic (***2ndPL***). We start with an example from [26]) as an instance of *while(index-set)*:

$\forall F_i provable(F_i)$

does not quantify over variable values (which would be 1stPL) but over a set of features. Generally, we encountered the need to write expressions such as

- **forall predicates satisfying some condition**
- **forall sorts satisfying some condition**

In both cases the sense of **forall** still needs to be defined. In its overall generality, these expressions are both beyond 1stPL. The meaning in all practical situations so far has, however, been that **condition** was a conjunction of a finite number of elementary condition i.e. predicates or a finite set of sub-sorts of the sort appearing in the scope of the "quantifier". In all these cases, the apparently quantified expression can be replaced by its finite expansion. Nonetheless, for reasons of comprehensibility for the reader (knowledge engineer or expert), the language should contain an expression for the apparent quantification. But for the very same reasons and furthermore to have a non-ambiguous deliverable for system design, the expression for the apparent quantification should be different from the logical quantifier. (This is indeed one of the hardest parts in the lin-

guistic data available from KADS papers, but also retrospectively from our own work, since one term - the quantifier - has been used with two absolutely different meanings; cf [22]'s argument about detection of conflict.)

Two control statements from the banking applications are as follows:

```
1. useall comparison(index)
        ... (goal statements about computations etc.)

2. if unknown(value(prev_yr))
```

**useall comparison** is expanded as "use all subsorts of the sort comparison, which are in the respective set **comparison*** of the actual **index**"

**unknown(value(prev_yr))** is expanded as attached-procedure predicate for **value** with a single in: - parameter for the instance of value and no out: - parameter. **unknown(value(prev_yr))** becomes true, when the attached-procedure predicate fails.

For a deeper understanding of the expressions in parentheses, the reader is referred to the subsequent presentation about the connection between the layers.

**Sequential task organisation** Although not directly requested in the KADS publications, fixed sequences of actions may play a role in certain forms of expertise. To cope with these, attached-procedure predicates can be used directly, i.e. the control structure of the procedural language can be made use of to formulate the sequence of actions and this can be naturally imbedded in the declarative environment.

# 5 Harmonics - Connection between layers

## *5.1 Domain and inference layer*

Role allocation of a domain layer entity to a metaclass is not an operation but rather the declaration that the domain layer entity is the role player or slot filler for a knowledge source. This leads to the introduction of (a possibly large set of) standard predicates **role_of**, each of which declares a domain entity to be an input or output metaclass:

```
role_of(name_of_domain_concept; Name_of_metaclass)
```

or, as for use in standard examples from systematic diagnosis ([2]):

```
role_of(patient_state; Complaint)
```

To deal with these at first sight 2nd order expressions of the form **predicate1(predicate2; ...)**, we demonstrate a possible 1stPL expansion on the banking example:

```
role_of(numeric_comparison;Pos_argument,Neg_argument)
```

is defined by

```
  role_of(X,Y;Pos_argument,Neg_argument)
& numeric_comparison(X,Y)
```

where **numeric_comparison** is defined by the axiom in the intermezzo Sec. 3.x. **Pos_argument** and **Neg_argument** are for use as input metaclass of the above introduced knowledge source **a_compute_difference. a_compute_difference** can operate on other domain entities by further **role_of** declarations such as

```
role_of(gain,loss;Pos_argument,Neg_argument)
```

defined, as above, by

```
  role_of(X,Y;Pos_argument,Neg_argument)
& gain(X) & loss(Y)
```

The semicolon between "actual parameters" (X,Y) from the domain and "formal parameters" of the knowledge source are only for better readability; they have no further meaning in the 1stPL base language. The whole expressions have to be understood as universally quantified for all variables. When in doubt, the quantifiers can be included in the description. The whole expanded description of the use of a domain concept in an inference then is as in the following example:

```
  numeric_comparison(X,Y)
& role_of(X,Y;Pos_argument,Neg_argument)
& a_compute_difference(in: Pos_argument,Neg_argument;
                       out:Numeric_difference)
  ...
```

and the respective **role_of** declaration for relating **Numeric_difference** to a domain concept. In other words: By virtue of their expansions, expressions such as

```
role_of(numeric_comparison;Pos_argument,Neg_argument)
```

are within 1stPL. By virtue of the distinct functions of "actual" domain and "formal" metaclass arguments they implement the explicit decoupling of domain and inference, as required in KADS. Tractable role_of-expressions can also be formulated for conclusions about the sort-heterarchy, inasmuch as they draw upon operations which are within the meet semilattice.

### *5.2 From domain via inference to task layer*

Inferences can be made immediate use of in task formulations such as

```
judgement_of_index(index,judgement)
←
useall comparison(index)
begin
    a_compute_difference(in:Numeric_comparison;out:Numeric_difference)
  & a_compare(in:Numeric_difference,Threshold;
              out:Relevance_of_difference)
  & a_evaluate_direction( ... )
end
```

which has the expansion

```
judgement_of_index(I,J)
←
    index(I)
  & useall comparison(I,C)
  & comparison_form(I,C,J,X,Y,Z)
  & role_of(X,Y;Pos_argument,Neg_argument)
  & role_of(Z;Difference)
  & a_compute(in: Pos_argument,Neg_argument;
              out: Difference)

  & role_of(Z,Tr;Argument,Threshold)
  & role_of(B,Relevance_of_argument)
  & a_compare(in: Argument,Threshold;
              out: Relevance_of_argument)
  ...
  & a_evaluate_direction( ... )
```

It means that the judgement of an index draws upon computation, test for threshold transgression, and eventually evaluation of the direction of the difference for all comparisons which are relevant for that index.
Short version and expansion play different roles with respect to each other: Firstly, the expansion establishes the 1stPL semantics of the short version. Secondly, the short version supports fast reading and comprehension of the overall structure of the task, whereas the expansion makes explicit a number of otherwise implicit relations.
(Note that there may be variables local to the task being solved; B, the temporary indication, whether the difference exceeds the threshold, is an example.)

We conclude that connection between layers can be reduced to 1stPL constructions, which can be used as a base representation onto which highly comprehensive constructs such as **useall(comparison(index))** can be mapped.

## 6 The Unfinished Status

The presented formalisation has been developed in parallel to an application project about evaluating the KADS method for its applicability in a standard software production environment of a group of banks. Only minor parts of the knowledge from a banking task were really dealt with systematically by our variation and increment of KADS. But those parts that had been formalized in precursor versions of the presented representation, could be taken by conventionally educated software engineers as input for system design.

The semantic foundation goes back to research projects LILOG ([17]) and ILFA ([18]) in the IBM Institute for Knowledge Based Systems. These two roots are currently being combined, which will in the near future also allow to have tools for editing and interpreting conceptual models.

An additional evaluation of the approach might be attempted by representing the elements of a library of interpretation models in the language.

## 7 Coda - Discussion

We have put emphasis in our presentation on a formal semantics for which there also exists a proof theory. Hence, we don't specify what can't be made executable.

The approach shall now be compared with the closely related one of [26].

There is a number of elegant suggestion in [26] concerning role limiting, lift-operations between layers, upward- and downward axioms between layers, modules, and import and export relations.

We claim that consequent use of sort expressions in domain layer and role-of construct for layer connection is an alternative solution for role limiting, which can be represented in 1stPL.

The lift concept may be more expressive than our 1stPL reducible role-of concept. But there seems to be a number of open questions about the semantics of lift. It is not clear, whether lift is a function, a (naming) relation, or a system of rewrite rules. In the second case, it might turn out identical with our role-of concept. In the case of rewrite rules, this may be hard for a common semantics with the other presented constructs.

upward and downward axioms seem to be comprised in the role-of, since we saw no need for a directionality in this point. If directionality turns out important, our attached-procedure predicates could be used.

We did not introduce modules in our presentation, though the usefullness of modules is principally highly appreciated. Technically they are available in the LILOG root of our approach. The import and export use of modules in [26] still seems to need some clarification. Import is either transitive, then there are some redundant imports in the examples. Or it is local, then the semantics has to be defined.

Semantics also has to be defined for a number of *a priori* non-logical constructs such as ***nonempty, exit, fail***, if a common semantics for the whole presented formalisation is to be achieved.

In conclusion: [26] offers a number of additional constructs which ease formulation of elements from conceptual models. The semantics of some of these still has to be thought over. On the contrary, our approach may be less expressive, but all constructs used can be based on 1stPL.

### *References*


[1] **Börger, E.** *Computability, Complexity, Logic* Amsterdam 1989 (North Holland)

[2] **Breuker, J.** *Model-Driven Knowledge Acquisition: Interpretation Models* Deliverable task A1, ESPRIT project 1098, University of Amsterdam, Dept. Social Science Informatics 2

[3] **deGreef, P., Breuker, J., Schreiber, G., and Wielemaker, J.** *StatCons: Knowledge Acquisition in a Complex Domain* Proc. ECAI 88, Munique, August 1988

[4] **Fodor, J.A.** *The Language of Thought* Thomas Y. Crowell, New York (1975)

[5] **Gallier, J.H.** *Logic for Computer Science* New York 1986 (Harper & Row)

[6] **Gurevich, Y.** *Logic and the Challenge of Computer Science* Boerger, E. (ed.); Trends in Theoretical Computer Science; Computer Science Press 1-57 (1988)

[7] **Hofstadter, D.R.** *Gödel, Escher, Bach: an Eternal Golden Braid* New York 1979 (Vintage books)

[8] **Horn, W.** *Diagnostic Decision Support Based on Generic Disease Descriptions and Detailed Anatomical Knowledge* Hunter, J., Cookson, J., and Wyatt, J. (Eds.); Proc. AIME 89, 2nd European Conference on Artificial Intelligence in Medicine, London, Aug. 1989, Springer Lect. Notes Med. Inf. 38 (1989)

[9] **Jansen-Winkeln, R.M.** *WASTL: An Approach to Knowledge Acquisition in the Natural Language Domain.* Boose, J., Gaines, B., and Linster, M. (eds.); Proc. European Knowledge Acquisition Workshop EKAW88, Bonn, July 88, GMD-Studien 143, St. Augustin 1988

[10] **Karbach, W., Voß, A., and Tong, X.** *Filling in the Knowledge Acquisition Gap: Via KADS' Models of Expertise to ZDEST-2' Expert System* Boose, J., Gaines, B., and Linster, M. (eds.); Proc. European Knowledge Acquisition Workshop EKAW88, Bonn, July 88, GMD-Studien 143, St. Augustin 1988

[11] **Klinker, G., Boyd, C., Dong, D., Maimann, J., McDermott, J., and Schnelbach, R.** *Building Expert Systems with KNACK* Boose, J. and Gaines, B. (eds.); Proc. 3rd Knowledge Acquisition for Knowledge-Based Systems Workshop; Banff (Canada) Nov. 1988

[12] **Linster, M.** *Integrating Acquisition, Representation, and Interpretation* Boose, J., Gaines, B., and Ganascia, J.G. (eds.); Proc. EKAW 89 3rd European Workshop on Knowledge Acquisition, Paris, July 1989 ISBN 2-90367763-8

[13] **Musen, M.A., Fagan, L.M., Combs, D.M., and Shortliffe, E.H.** *Use of a Domain Model to Drive an Interactive Knowledge-Editing Tool* Int. J. Man-Machine Studies **26**, (1987), 105-121

[14] **Nebel, B.** *Terminological Reasoning is Inherently Intractable* IWBS Report 82, IBM Germany, Scientific Center, PO Box 800880, D-7000 Stuttgart 80, F.R.Germany

[15] **Nordbo, I., Vestli, M., and Solvberg, I.** *METAMETH Methodology for Knowledge Engineering* Boose, J., Gaines, B., and Ganascia, J.G. (eds.); Proc. EKAW 89 3rd European Workshop on Knowledge Acquisition, Paris, July 1989 ISBN 2-90367763-8

[16] **O'Neil, M., Glowinski, A., and Fox, J.** *A Symbolic Theory of Decision-Making Applied to Several Medical Tasks* Hunter, J., Cookson, J., and Wyatt, J. (Eds.); Proc. AIME 89, 2nd European Conference on Artificial Intelligence in Medicine, London, Aug. 1989, Springer Lect. Notes Med. Inf. 38 (1989)

[17] **Pletat, U. and von Luck, K.** *Knowledge Representation in LILOG* (to appear in) Proc. Workshop Order-Sorted Knowledge Represenation and Inference Systems, Springer Lecture Notes in Artificial Intelligence

[18] **Schönfeld, W.** *Properties and Actions* (to appear in) Proc. IBM Symposium on Natural Language and Logic, Hamburg, May 1989, Springer Lecture Notes Computer Science

[19] **Schmitt, P.H. and Wernecke, W.** *Tableau Calculus for Order Sorted Logic* (to appear in) Proc. Workshop Order-Sorted Knowledge Represenation and Inference Systems, Springer Lecture Notes in Artificial Intelligence

[20] **Schreiber, G., Breuker, J., Bredeweg, B., and Wielinga, B.** *Modelling in KBS Development* Boose, J., Gaines, B., and Linster, M. (eds.); Proc. European Knowledge Acquisition Workshop EKAW88, Bonn, July 88, GMD-Studien 143, St. Augustin 1988

[21] **Schreiber, G.** personal communication October 1988

[22] **Shaw, M. and Gaines, B.** *A Methodology for Recognizing Consensus, Correspondence, Conflict, and Contrast in a Knowledge Acquisition System* Boose, J. and Gaines, B. (eds.); Proc. 3rd Knowledge Acquisition for Knowledge-Based Systems Workshop; Banff (Canada) Nov. 1988

[23] **Smeets, R.P.A.M., Talmon, J.L., van der Lugt, P.J.M., and Schijven, R.A.J.** *The Development of a Knowledge System for Surveillance of Anti-Epileptic Medication* Hunter, J., Cookson, J., and Wyatt, J. (Eds.); Proc. AIME 89, 2nd European Conference on Artificial Intelligence in Medicine, London, Aug. 1989, Springer Lect. Notes Med. Inf. 38 (1989)

[24] **Walther, Ch.** *A Many Sorted Calculus based on Resolution and Paramodulation* Proc. 8th IJCAI, Karslruhe, 1983

[25] **Wielinga, B.J., Bredeweg, B., and Breuker, J.A.** *Knowledge Acquisition for Expert Systems* Nossum, T. (ed.); Advanced Topics in Artificial Intelligence; Lecture Notes in Artificial Intelligence 345, Berlin 1987 (Springer), 96 - 124

**[26] Wielinga, B.J., Akkermans, H., Schreiber, G., and Balder, J.** *A Knowledge Acquisition Perspective on Knowledge-Level Models* Proc. 4th Knowledge Acquisition for Knowledge-Based Systems Workshop; Banff (Canada) Oct. 1989

**[27] Xie, S., Dumaresq, D.F., and Winne, P.H.** *PRED: An Interface Development Tool for Editing Primitives in Knowledge Representation Languages* Boose, J. and Gaines, B. (eds.); Proc. 3rd Knowledge Acquisition for Knowledge-Based Systems Workshop; Banff (Canada) Nov. 1988

# *Acknowledgments*

Udo Pletat and Wolfgang Schönfeld helped me to understand the expressiveness and foundation of 1stPL variants used in this presentation.
Wolfgang Wernecke and Astrid Schmücker gave important comments on a former draft of this text.
Egon Boerger gave valuable advice about the dynamic algebra aspects of procedural attachment.
The anonymous reviewers of a former version precisely pointed out vague or inclear formulations and (hopefully) enabled me to put things clearer now.
Peter Curwiel of Bolesian taught me the importance of interpretation models.
P. Marinopoulos, H.J. Christ, W. Hoppe, and R. Weigelt of the Betriebswirtschaftliche Institut der Deutschen Kreditgenossenschaften (BIK) provided permanent critical discussion of the usefulness of the evolving concept.
Mark Beers patiently helped going through a number of iterations of preparing the final text and gave much advice about how to write things in English.
***Thanks to all of them.***