Fifth European Knowledge Acquisition for Knowledge-Based Systems Workshop

EKAW91

May 20th - May 24th 1991

Crieff Hydro, Scotland

Workshop Proceedings

(Pre-print)

Colloquium on Knowledge Acquisition for Knowledge-Based Systems

May 20th 1991

Strathclyde University, Scotland

Machine Learning, Knowledge Base Refinement and Knowledge Acquisition: Systems which Cope with Incomplete and Inconsistent Knowledge Bases

D Sleeman, Susan M Craw[†], Ian G Ellery[§],
Sunil Sharma & Peter Edwards

Department of Computing Science
King's College
University of Aberdeen
Aberdeen, AB9 2UB
Scotland, UK

Abstract

There are clearly three distinct phases in the production of a knowledge-based system, namely: knowledge acquisition, the encoding of the knowledge in a computer-based system, and its refinement. To date these phases have often been undertaken independently of each other; and the latter two phases are only now beginning to benefit from computer support. As knowledge bases (KBs) get larger and more complex, it is important that tools be developed to help detect errors and omissions, and then correct them. We need to develop a range of techniques for coping with incomplete KBs, which must be incorporated into (future) intelligent systems. Over the last three years a number of systems have been built at Aberdeen which deal with KBs which are both incomplete and inconsistent, namely: KRUST, CREATOR and REFINER. These systems are outlined,we discuss how we intend developing them further and describe their use as knowledge acquisition/knowledge refinement tools.

1 Introduction

There are clearly three distinct phases in the production of a knowledge-based system, namely: knowledge acquisition, the encoding of the knowledge in a computer-based system, and its refinement. To date these phases have often been undertaken independently of each other; and the latter two phases are only now beginning to benefit

---

† School of Computer and Mathematical Sciences, Robert Gordon's Institute of Technology, Aberdeen, AB1 1HG.

§ Computing Centre, University of East Anglia, Norwich, NR4 7JT.

from computer support. Examples of computer systems which support knowledge acquisition are ETS/AQUINAS [1], SHELLEY [2] (based on the KADS methodology), and ProtoKEW [3]. In recent years, there have been important attempts to combine these phases within single systems; this work has been undertaken primarily by McDermott and his co-workers [4].

As knowledge bases (KBs) get larger and more complex, it is important that tools be developed to help detect errors and omissions [5], and then correct them. We need to develop a range of techniques for coping with incomplete KBs, which must be incorporated into (future) intelligent systems. It should be noted, in passing, that humans have a range of techniques for coping with such situations including: adding plausible assumptions, using analogies [6, 7], and negative inferences.

Over the last three years several systems have been built at Aberdeen which deal with KBs which are both incomplete and inconsistent; for a more general overview of recent work in knowledge refinement and "apprentice systems", see [8]. The systems developed at Aberdeen are outlined in the following sections.

2 KRUST: Assists the Designer of an Expert System to Refine its Knowledge Base

KRUST is an automated KB refinement system which removes the need for the expert and knowledge engineer to perform tedious refinement tasks. It uses a *range* of knowledge sources to automate KB refinement: statistical meta-knowledge, heuristics and task-solution pairs. It is novel for a refinement system to generate *many* refinements for the KB and postpone its choice of a suitable refinement until it has evidence of their quality. Testing has shown that KRUST is able to generate and manage these multiple refinements, [9,10]. For a simple wine advising KB, KRUST's filtering techniques restrain the process within manageable bounds. We are able to apply coarse testing to a large number of refinements, so that finer testing is only applied to a significantly reduced number of KBs.

KRUST is presented with a single wrongly solved task, the expert's solution and the KB's solution. It generates all possible combinations of "fixes", and then eliminates those which involve making changes to rules which are thought to be strong; statistically generated meta-knowledge is used to make this decision. The next step is to build all the KBs which are consistent with the remaining "fixes". Filtering rejects those KBs which are inconsistent with the expert's solution for the training task and the known solutions for a set of "chestnut" cases; if a KB wrongly answers one of these crucial cases then this is sufficient evidence that it is unsuitable. The remaining KBs are subjected to a more detailed analysis which ranks them, and the best KB is recommended to the user. The components of KRUST are shown in Figure 1.

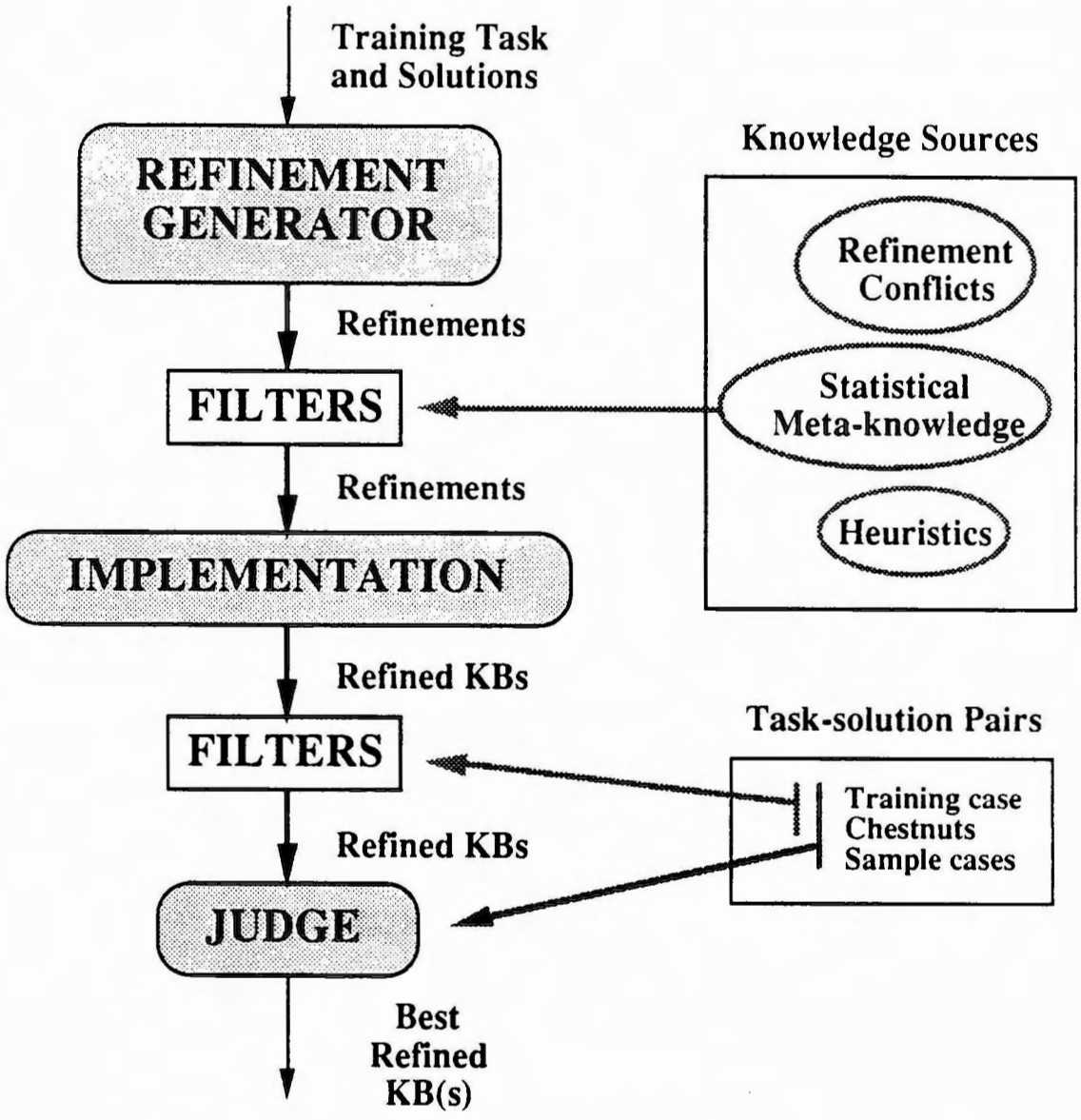


Figure 1 : The KRUST System.

In summary, KRUST generates many refinements, rejecting them only once blame is attributed to them. Thus, KRUST's philosophy does not pre-judge refinement quality, as many refinement systems seem to do by generating a *single* refinement [11].

Additional work planned with KRUST includes:

- Refinement of larger KBs in more complex domains. Segmenting complex KBs into modular sub-theories to allow KRUST to refine the sub-theories individually, thus improving efficiency.
- Creating artifacts given the design for an analogous device. KRUST's speculative approach may be relevant to general design exercises.

- Improving KB completeness as well as consistency. KRUST's approach could assist an acquisition tool in the integration of new rules. Currently, KRUST creates only "crude" new rules; addressing this problem has been the focus of Infer* and CREATOR, described in the next section.

3 CREATOR: Creation of Possible New Rules for an Incomplete Knowledge Base

If a KB is missing one (or a few) rules, then for a given input, an incorrect answer will normally be reached. The aim of CREATOR [12] is to discover the missing rule(s), given the correct answer (that given by the domain expert) and the original inputs to the KB. Often, a selection of rules will be found and presented to the expert for analysis. We believe that it is far simpler for an expert to select a new rule from a small number of plausible choices, than to attempt to design a new rule for himself.

CREATOR discovers new rules by searching *bidirectionally* from the inputs and the expert's answer, and then comparing pairs of states from both searches. (These states are just sets of facts, i.e. sets of working memory elements.) If two states are "close enough" then a new rule to connect them can be inferred, thus completing the search path. The idea of bidirectional search in this way was first used in the Infer* system [13]. This system was designed to discover new *mal-* rules for student protocols, and was restricted to the domain of high school algebra. One of the problems of Infer* was that far too many mal-rules were suggested, and so some way of filtering the set of generated rules was required.

CREATOR uses heuristics in two ways to limit the number of rules produced. If, when two states are compared, there are too many facts which cannot be matched, then it is assumed that the two states are not sufficiently close, and so no rule is formed. (The exact way this is done depends on various domain specific parameters [12].) After many new rules have been created, other heuristics are used to filter out implausible new rules. These heuristics are domain specific, and are built *automatically* from the initial KB by CREATOR. This automatic construction is essential for a system of this type as the expert cannot be expected to provide these heuristics; a refinement system is supposed to ease the task of the KB builder, not make it harder.

The main loop of the CREATOR system is shown in Figure 2. First the bidirectional search is performed, then the inference of new rules. After the heuristics filter out the implausible rules, the remaining rules are generalised. This generalization is necessary, as the new rules are being produced from only a *single* failed example.

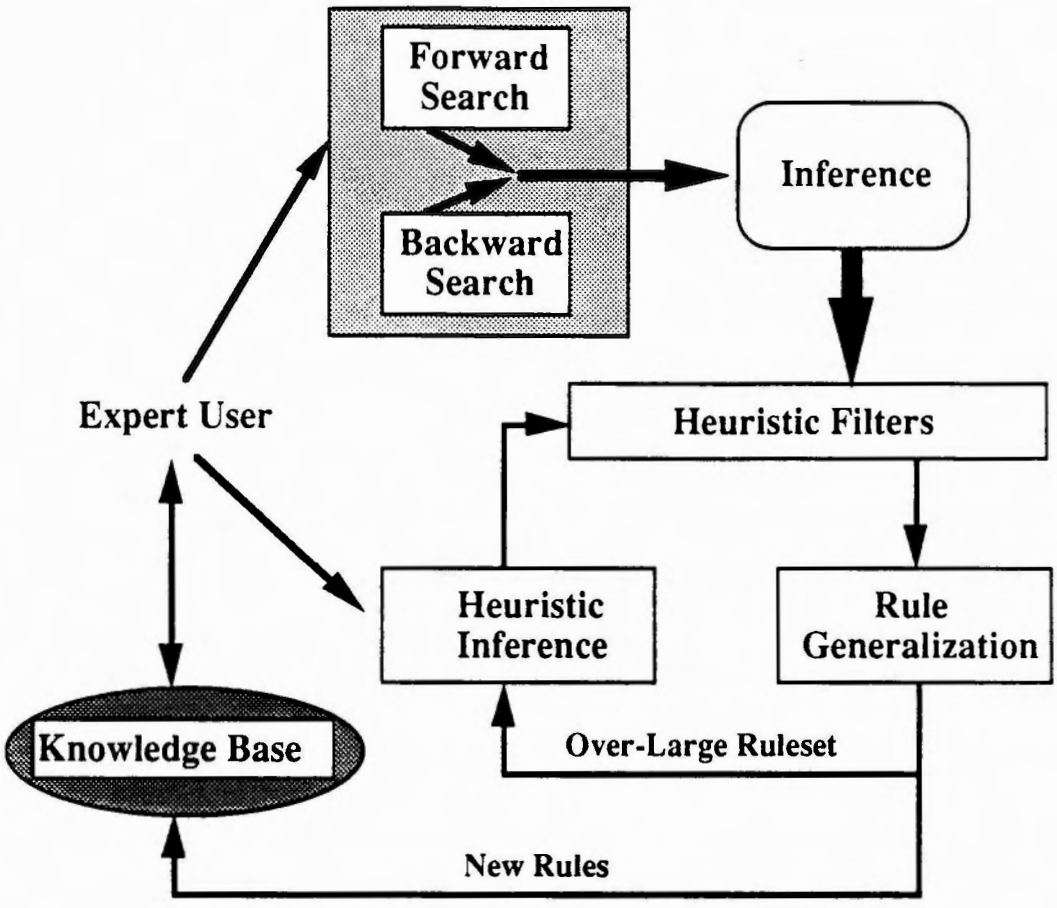


Figure 2 : A Schematic Diagram of the CREATOR System.

Since the filter heuristics were built automatically, they too may need refinement. The set of heuristics is refined by having the expert accept or reject each of the rules generated by CREATOR, from which the system builds additional heuristics. Those rules which are accepted by the expert are automatically incorporated into the KB.

We plan to integrate CREATOR and KRUST into a coherent knowledge refinement system, attached to an expert system shell and capable of refining sizable real world KBs. This would require extensions to various parts of the system, for example to cope with certainty factors and multiple conclusions in rules. There are also some alterations required for efficiency purposes. These and other changes are discussed fully in [12].

4 REFINER: Detects and Removes Inconsistencies in Cases

REFINER is a system which helps the expert refine a KB consisting of a set of cases and their assigned diagnoses or, more generally, classifications. (Kolodner [14] discusses advantages of case-based reasoning over more traditional approaches to Knowledge Elicitation). Further, REFINER is told whether the expert believes each case to be a prototypical or non-prototypical example of the class. By using background knowledge

stored in a semantic network, REFINER infers descriptions of prototypes and non-prototypes for each of the classes. (The description for the non-prototype of class A is a generalisation of the prototype for class A).

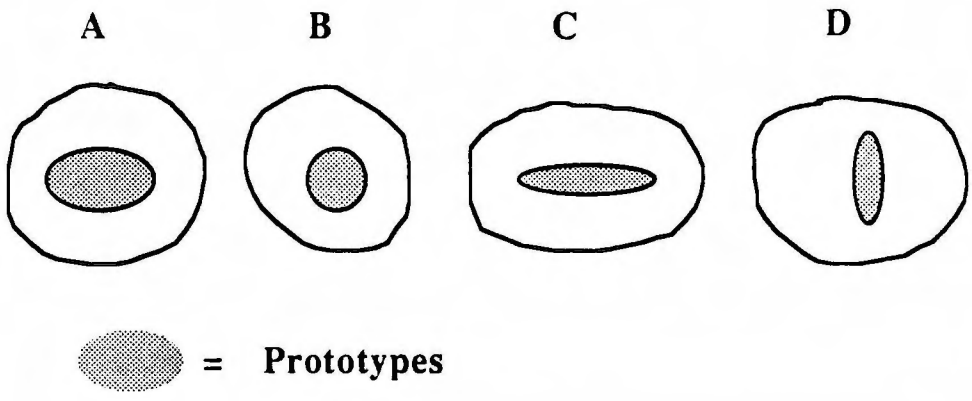


Figure 3 : Showing four non-overlapping classes in REFINER.

Figure 3 gives a graphical view of a KB which contains four classes A, B, C, and D; in this case the KB is consistent. If the KB had been inconsistent then at least one example would be classified differently from the class specified by the expert (this includes the system being *unable* to classify the case).

4.1 Assumptions

The Assumptions which underlie this work include:

- Each (disease) class has some distinct features.
- A non-prototype has at least one feature which is a variant of a feature found in a prototype; i.e., a non-prototype must not have a set of completely unique features.

4.2 Simplified Algorithm

1. Present a prototypical example of each class; REFINER infers a set of Class-Distinguishing-Features (CDF) for each class.

2. Presentation of Non-Prototypes generalises the appropriate CDF. Generalisation can be performed on both numerical ranges (e.g. 80-84 & 84-86 → 80-86) and symbolic entities (e.g. arm and leg → limb - this uses the semantic network mentioned earlier).

3. When encountering a case of D, as well as updating D's CDF also update the CDFs for A, B, and C to remove all overlapping features (i.e. all the features in D) as these can no longer be distinctive for classes A, B, and C.

4. Detecting inconsistencies and suggesting how to remove them (this is discussed later).

4.3 A Simple Example

PROTOTYPE: DISEASE-A(P#24) =

AGE = 50-54 and
DBP = 85-89 and
ADDITIONAL-DISEASE = DIABETES

PROTOTYPE: DISEASE-B(P#85) =

AGE = 50-54 and
DBP = 105-109 and
ADDITIONAL-DISEASE = OBESITY

These are to be read as:

- Case P24 is a prototypical example of class A where the patient's age is in the range 50-54, the DBP (Diastolic Blood Pressure) is in the range 85-89 and the patient also has Diabetes.
- Case P85 is a prototypical example of class B where the patient's age is in the range 50-54, the DBP is in the range 105-109 and the patient additionally suffers from Obesity.

From this information REFINER rules out age as being a distinctive feature of either class (as the same range of values is present in both cases) and infers the following Class Distinguishing Features (CDFs):

CDF (DISEASE-A) =

DBP = 85-89
and ADDITIONAL-DISEASE = DIABETES

CDF (DISEASE-B) =

DBP = 105-109 and
ADDITIONAL-DISEASE = OBESITY

4.4 Detection and Removal of Inconsistencies

An inconsistency occurs when a case is classified differently by the system and by the expert. The reason for the misclassification is possibly because features of the class have been removed from the appropriate CDF. The example above gives some clue as to how impoverished CDFs occur. Any feature in a case identified as D, causes all those features to be removed from the CDFs for classes A, B and C. If REFINER is unable to "correctly" match an example which the expert asserts is an A, then it reconsiders all the cases (of other classes) which forced it to remove a feature from A's CDF; these cases are all candidates for investigation. Specifically, REFINER suggests that the following types of modifications should be considered:

1. Specific case(s) be reclassified.
2. Edit case(s), e.g., the values of particular attribute(s).
3. Add new descriptor(s) to all cases.
4. Shelve case(s).
5. Modify the background domain knowledge.

These mechanisms are discussed in greater detail in references [15, 16].

4.5 Summary

REFINER does a great deal of processing of examples, and then presents the expert with the cases which cause the inconsistency to occur. Once confronted with highly focused information humans are often good at suggesting ways of overcoming the discrepancy. Thus, REFINER seems to have many of the desirable features of an aide, and nicely illustrates the theme of co-operative problem-solving.

4.5.1 Is REFINER a Knowledge Elicitation Tool ?

The problem of getting expert users to articulate their domain knowledge is well documented. We would like to argue that REFINER allows this information to be obtained in a fairly painless way. In fact, the actual cases used in the above examples are taken from the records of patients receiving long term therapy for Hypertension, and were taken directly from the hospital database and transformed by us into a format suitable for REFINER. Further, we would argue that asking the expert to classify cases and to specify whether each case is a Prototype or a Non-Prototype is again something which most experts can readily do, and yet it provides the system with considerable valuable information.

4.5.2 Planned Extensions to REFINER

• Extend the distinctions elicited between prototypes and non-prototypes to finer distinctions, such as "near prototype", "extreme non-prototype" etc. This information should enable additional inconsistencies to be determined.

• Encourage the expert to say which classes he sometimes gets confused between (for which he finds a differential diagnosis difficult). For example, if he says that non-prototypes of classes A and D are hard to distinguish, then the descriptions of these cases should have considerable overlap in the REFINER representation. If they do not, REFINER might, for example, systematically check the features of the two "difficult" cases to determine whether some piece of background knowledge was missing. For example, the system might check to see if the two drugs being used in the cases are related. Additionally, one might ask the expert why he thought the two cases were difficult to differentiate, and then use this information directly.

• Determine the effect of expanding ranges of values; use background knowledge to generalize inferred class descriptors, for example to test whether it is possible to replace *apple* by *fruit*.

5 Postscript: Machine Discovery Systems

We also have a long term interest in machine discovery systems [17], and have hypothesized that they can be profitably viewed as systems which involve the refinement of (multiple) knowledge bases. Detection of a concept which is not associated with any other concepts, leads to investigation in an effort to create such associations (and perhaps additional unexpected associations.) Several systems which embody these perspectives have been built, and we plan shortly to use such systems with tasks which scientists are still attempting to resolve; that is, domains in which scientists are still in the process of refining appropriate theories.

Acknowledgements

Sunil Sharma and Ian Ellery acknowledge the financial support provided by the Science and Engineering Research Council in the form of SERC studentships, which led to the development of the REFINER and CREATOR systems. Our work on machine discovery is supported by the EEC's Science Program, contract SC1 0048 C(H).

References

1. Boose JH, Bradshaw JM. Expertise Transfer and Complex Problems: Using AQUINAS as a Knowledge Acquisition Workbench for Knowledge-Based Systems. Int J Man-Machine Studies 1987; 3-28

2. Anjewierden A. Shelley User's Guide. Technical Report UvA-C*-SUG-1.00, University of Amsterdam, 1990

3. Shadbolt N, Wielinga B. Knowledge-Based Knowledge Acquisition: The Next Generation of Support Tools. In: Wielinga B et al. (eds) Current Trends in Knowledge Acquisition. IOS Press, Amsterdam, 1990, pp 313-338

4. Marcus S. Automating Knowledge Acquisition for Expert Systems. Kluwer, Boston, 1988

5. Suwa M, Scott AC, Shortliffe EH. Completeness and Consistency in a Rule-Based System. In: Buchanan BG, Shortliffe EH (eds) Rule-Based Expert Systems. Addison-Wesley, Menlo Park, 1984, pp 159-170

6. Gentner D. Structure-mapping: A Theoretical Framework for Analogy. Cognitive Science 1983; 7:155-170

7. Carbonell JG. Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition. In: Michalski RS, Carbonell JG, Mitchell TM (eds) Machine Learning: An Artificial Intelligence Approach, Volume II. Morgan Kaufmann, Palo Alto, 1986, pp 371-392

8. Wilkins DC. Knowledge Base Refinement Using Apprenticeship Learning Techniques. In: Smith RG, Mitchell TM (eds) Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88). Morgan Kaufmann, Palo Alto, 1988, pp 646-651

9. Craw SM, Sleeman D. Automating the Refinement of Knowledge-Based Systems. In: Aiello LC (ed) Proceedings of the Ninth European Conference on Artificial Intelligence (ECAI90). Pitman, London, 1990, pp 167-172

10. Craw SM. Automating the Refinement of Knowledge-Based Systems. PhD Thesis, Computing Science Department, University of Aberdeen, 1991

11. Craw SM, Sleeman D. The Flexibility of Speculative Refinement. In: Birbaum L, Collins G. (eds) Machine Learning: Proceedings of the Eighth International Workshop. Morgan Kaufmann, San Mateo, CA. (to appear)

12. Ellery IG. The Creation of Missing Rules for Knowledge-Based Systems. PhD Thesis, Computing Science Department, University of Aberdeen, 1991

13. Sleeman D, Hirsh HB, Ellery IG, Kim I-Y. Expanding an Incomplete Domain Theory: Two Case Studies. Machine Learning Journal 1990; 5:11-37

14. Kolodner JL. Maintaining Organization in a Dynamic Long-Term Memory. Cognitive Science 1983; 7:243-280

15. Sharma S, Sleeman D. REFINER: A Case-Based Differential Diagnosis Aide for Knowledge Acquisition and Knowledge Refinement. In: Sleeman D (ed) Proceedings of the European Working Session on Learning (EWSL-88). Pitman, London, 1988, pp 201-210

16. Sharma S. Case-Based Knowledge Acquisition and Refinement. PhD Thesis, Computing Science Department, University of Aberdeen, 1988

17. Sleeman D, Stacey MK, Edwards P, Gray NAB. An Architecture for Theory-Driven Scientific Discovery. In: Morik K (ed) Proceedings of the European Working Session on Learning (EWSL-89). Pitman, London, 1989, pp 11-23

The KADS Framework for Modelling Expertise*

A.Th. Schreiber B.J. Wielinga J.A. Breuker

Department of Social Science Informatics, University of Amsterdam
Roetersstraat 15, NL-1018 WB Amsterdam, The Netherlands
Tel: +31 20 525 6789; Telecopier: +31 20 525 6896
Electronic mail: schreiber@swi.psy.uva.nl

Abstract

This paper discusses the KADS approach to knowledge modelling. In KADS, the development of a knowledge-based system (KBS) is viewed as a modelling activity. A KBS is not a container filled with knowledge extracted from an expert, but an operational model that exhibits some desired behaviour that can be observed in terms of real-world phenomena. The KADS framework for modelling expertise is described in some detail. The reusability of models of expertise through "interpretation models" is discussed. Relevant design/implementation issues are addressed. The KADS framework is compared to related approaches. The approach is illustrated throughout the paper with examples in a domain of troubleshooting audio equipment.

1 Introduction

In the knowledge acquisition process the knowledge that a knowledge-based system (KBS) needs in order to perform a task, is defined in such a way that a computer program can represent and adequately use that knowledge. Knowledge acquisition involves in our view at least the following activities: *eliciting* the knowledge in an informal — usually verbal — form, *interpreting* the elicited data using some conceptual framework, and *formalising* the conceptualisations in such way that the program can use the knowledge. In this paper we will mainly focus on the interpretation and formalisation activities in knowledge acquisition. Elicitation techniques have been the subject of a number of recent papers and their role in the knowledge acquisition process is reasonably well understood [Meyer & Booker, 1991; Neale, 1988; Breuker & Wielinga, 1987].

*This article is a shortened version of an article submitted to *Knowledge Acquisition*. The production of this article was supported by the KADS-II project. This project is partially funded by the ESPRIT Programme of the Commission of the European Communities as project number 5248. The partners in the project are Cap Gemini Innovation (France), Cap Gemini Logic (Sweden), Netherlands Energy Research Foundation ECN (The Netherlands), ENTEL SA (Spain), Lloyd's Register (United Kingdom), Swedish Institute of Computer Science (Sweden), Siemens AG (Germany), Touche Ross MC (United Kingdom), University of Amsterdam (The Netherlands) and Free University of Brussels (Belgium). Disclaimer: this paper reflects the opinions of the authors and not necessarily those of the consortium.

Traditionally the knowledge acquisition process has been viewed as a process of extracting knowledge from a human expert and transferring the extracted knowledge into the KBS. In practice this often means that the expert is asked what rules are applicable in a certain problem situation and the knowledge engineer translates the natural language formulation of that rule into the appropriate format. Several authors [Hayward et al., 1987; Morik, 1989] have pointed out that the transfer-view of knowledge acquisition is only applicable in very few cases. The expert, the knowledge engineer and the KBS should share a common view on the problem solving process and a common vocabulary in order to make knowledge transfer a viable way of knowledge acquisition. If the expert looks in a different way than the knowledge engineer at the problem or the domain, asking for rules or similar knowledge structures and translating them into the knowledge representation formalism of the system, does not work.

A different view on knowledge acquisition is that of a modelling activity. A KBS is not a container filled with knowledge extracted from an expert, but an operational model that exhibits some desired behaviour observed or specified in terms of real-world phenomena. The use of models is a means of coping with the complexity of the development process. A model reflects, through abstraction of detail, selected characteristics of the empirical system in the real world that it stands for [DeMarco, 1982].

Constructing a KBS is seen as building a computational model of desired behaviour. Often, this desired behaviour coincides with some behaviour as exhibited by an expert. If one wants to construct a KBS that performs medical diagnosis, the behaviour of a physician in asking questions and explaining the problem the problem of a patient may exemplify this intended behaviour of the KBS. However, often the introduction of information technology involves new distributions of functions and roles of agents. This implies that a KBS is hardly ever the functional equivalent of an expert. The KBS may perform functions which are not part of the experts repertory. Also, the underlying reasoning process which should enable these functions, is not explicit. Knowledge, principles and methods may be documented in a domain, but these are aimed at a human interpreter and are not explicit descriptions of how to solve problems in a mechanical way. Also, there is an inherent difference between the inference capabilities of machines and humans. For example, in an experiment in a domain of configuring moulds [Barthélemy et al., 1988] a decision was made to generate all possible solutions instead of the small set generated by experts. The decision was guided by the fact that for a machine it presents no problem to store a large number of hypotheses in short-term memory, whereas for humans this is impossible.

So, in the modelling view knowledge acquisition essentially is a constructive process in which the knowledge engineer can use all sorts of data about the behaviour of the expert, but in which the ultimate design decisions have to be made by the knowledge engineer in a constructive way. In this sense knowledge engineering is similar to other design tasks: the real world only provides certain constraints on what the artefact should provide in terms of functionality, the designer will have to aggregate the bits and pieces into a coherent system.

2 Intermediate Models in the Knowledge Engineering Process

The construction of a knowledge-based system is a complex process. It can be viewed as a search through a large space of knowledge-engineering methods, techniques and tools. Numerous choices have to be made with regard to elicitation, conceptualisation and formalisation. Knowledge engineers are thus faced with a jungle of possibilities and find it difficult to navigate through this space.

The knowledge-engineering space of choices and tools can to some extent be controlled by the introduction of a number of *intermediate models*. Each model emphasises certain aspects of the system to be built and abstracts from others. Intermediate models provide a decomposition of knowledge-engineering tasks: while building one model, the knowledge engineer can temporarily neglect certain other aspects. The complexity of the knowledge-engineering process is thus reduced through a divide-and-conquer strategy.

In this section we discuss two intermediate models, namely (i) the model of expertise and (ii) the design model. Others models distinguished in KADS are the organisational model, the application model, the task model and the model of cooperation [Wielinga et al., 1991].

We use the term *knowledge engineering* in a broad sense to refer to the overall process of KBS construction (i.e. the construction of all these models and the artefact) and the term *knowledge acquisition* in a more restricted sense to refer to those parts of this construction process that are concerned with the information about the actual problem solving process. The scope of the present article is limited to the knowledge acquisition aspects. Other knowledge engineering aspects are only briefly addressed.

2.1 Model of Expertise

Building a model of expertise is a central activity in the process of KBS construction. It distinguishes KBS development from conventional system development. Its goal is to specify the problem solving expertise required to perform the problems solving tasks assigned to the system.

One can take two different perspectives on modelling the expertise required from a system. A first perspective – the one that is often taken in AI – is to focus on the computational techniques and the representational structures (e.g. rules, frames) that will provide the basis of the implemented system. A second perspective focuses on the behaviour that the system should display and on the types of knowledge that are involved in generating such behaviour, abstracting from the details of how the reasoning is actually realised in the implementation. These two perspectives correspond to the distinction Newell [Newell, 1982] makes between respectively the *symbol level* and the *knowledge level*.

We take the second perspective and view the model of expertise as being a knowledge-level model. The model of expertise specifies the desired problem solving behaviour for a target KBS through an extensive categorisation of the knowledge required to generate this behaviour. The model thus fulfills the role of a *functional specification* of the problem solving part of the artefact. As stated previously, it is not a cognitive model of the human expert. Although the construction of the model of expertise is usually guided by an analysis of expert behaviour, it is biased to what the target system should and can do.

In modelling expertise we abstract from those sub-tasks that specify some form of cooperation with the user. For example, in the audio domain we could identify two tasks that require such interactions: *carrying out a test* and *implementing a proposed reconfiguration or remedy*. In the model of expertise, such interaction or *transfer* tasks are specified more or less as a black box (see Section 3.3). The detailed study of the nature of these transfer tasks is the subject of the modelling of cooperation. The model of cooperation is outside the scope of this paper. For more details about the model of cooperation, see [de Greef & Breuker, 1989; de Greef et al., 1988a]).

The framework for modelling expertise is discussed in detail in Section 3. The reuse of (partial) models of expertise is addressed in Section 4.

2.2 Design Model

The description of the computational and representational techniques that the artefact should use to realise the specified behaviour is not part of the model of expertise. These techniques are specified as separate *design decisions* in a design model. In building a design model, the knowledge engineer takes external requirements such as speed, hardware and software into account. Although there are dependencies between model of expertise specifications on the one hand and design decisions on the other hand, we have experienced that building a model of expertise without having to worry about system requirements makes life easier for the knowledge engineer.

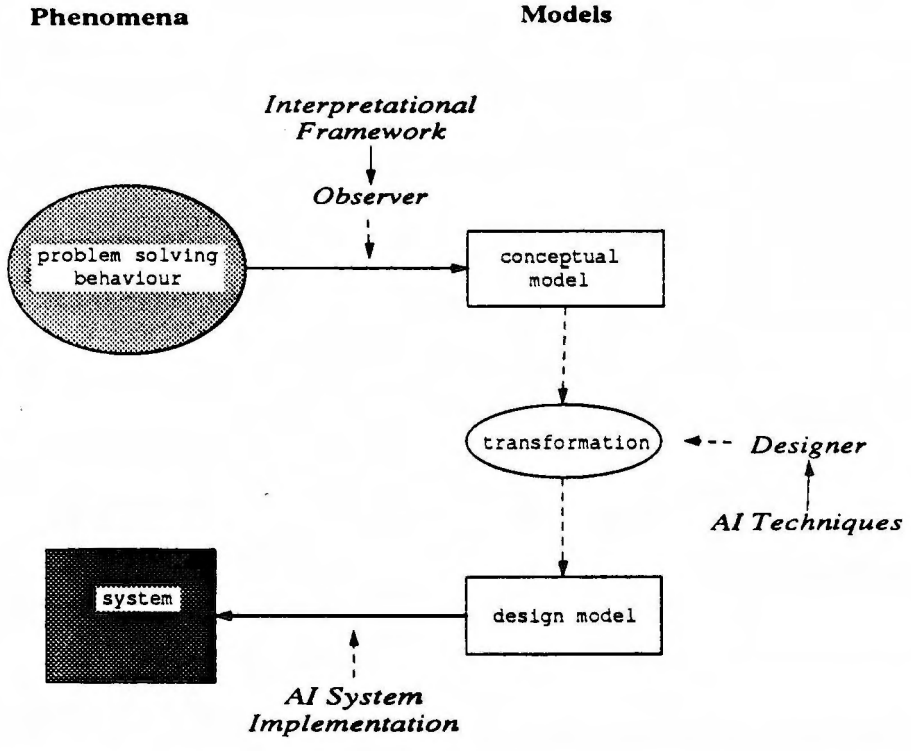


Figure 1: Role of the model of expertise and the design model in the knowledge acquisition process

Fig. 1 summarises the different roles which the model of expertise and the design model play in the knowledge engineering process. An observer (knowledge engineer) constructs a knowledge-level model of the artefact – the model of expertise – 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 classes of tasks or task-domains. The model of expertise is real-world oriented in the sense that it is phrased in real-world terminology and can thus be used as a communication vehicle between knowledge engineer and expert. The model of expertise does not take detailed constraints with respect to the artefact into account. The *design model* is a *symbol-level* model of the artefact. It describes how the model of expertise is realised with particular computational and representational techniques.

Design decisions relevant for the operationalisation of the model of expertise are discussed in Section 5.

3 A Framework for Modelling Expertise

The major challenge for any modelling approach to KBS construction is to find an adequate answer to the question of how to model expertise. It is this aspect of the system that distinguishes KBS development from the development of conventional systems. As discussed previously, we require of the resulting model of expertise that it is independent of a particular implementation. In this section a framework for modelling expertise is outlined. Slightly different versions of this KADS approach to modelling expertise (usually called the "four layer model") have been presented in [Wielinga & Breuker, 1986; Hayward et al., 1987; Schreiber et al., 1988; Breuker & Wielinga, 1989]

Two basic premises underly the ideas presented here. First, we assume that it is possible and useful to distinguish between several generic types of knowledge according to different *roles* that knowledge can play in reasoning processes. Second, we assume that these types of knowledge can be organised in several *layers*, which have only limited interaction. A first distinction that is often made is the distinction between *domain knowledge* and *control knowledge*. Here we will take such a separation of knowledge in two layers one step further, and will argue for a refined distinction of different types of control knowledge at three levels.

The categories in which the expertise can be analysed and described are based on epistemological distinctions: they contain different types of knowledge. We distinguish between:

1. Static knowledge describing a declarative *theory* of the application domain (domain knowledge).
2. Knowledge of different *types of inferences* that can be made in this theory (first type of control knowledge).
3. Knowledge representing *elementary tasks* (second type of control knowledge).
4. *Strategic knowledge* (third type of control knowledge)

Each of these categories of knowledge is described at a separate level. The separation reflects different ways in which the knowledge can be viewed and used. In the following

sections each of the four categories of knowledge distinguished in KADS is discussed in more detail.

The distinction between different types of knowledge is not new. Several authors have reported ideas which pertain to the separation of domain and control knowledge, and ways to increase the flexibility of control in expert systems. The work of Davis [Davis, 1980] introduced explicit control knowledge as a means to control inference processes in a flexible way. In the NEOMYCIN system [Clancey, 1985a] different functions of knowledge are explicated by separating domain knowledge and control knowledge and introducing a explicit description of the strategies that a system uses. Pople [Pople, 1982] has stressed the problem of the right task formulation. He considers it to be a fundamental challenge for AI research to model the control aspects of the reasoning process of expert diagnosticians which determines the optimal configuration of tasks to perform in order to solve a problem.

3.1 Domain Knowledge

The domain knowledge embodies the *conceptualisation* of a domain for a particular application in the form of a *domain theory*. The ontological primitives that we use to describe domain theories are: concepts, properties, and two types of relations.

Concept

Concepts are the central objects in the domain theory. A concept is identified through its name (e.g. `amplifier`).

Property/Value

Concepts can have *properties*. Properties are defined through their name and a description of the values that the property can take. For example, `amplifier` has a property `power` with as possible values `on/off`.

Relation between concepts

A first type of relation is the relation between concepts, for example `amplifier is-a component`. The most common relations of this type are the sub-class relation and the part-of relation. Several variants of these two relations exist, each with its own semantics. This type of relation is sometimes called a *structural relation*.

Relation between property expressions

A second type of relation is the relation between expressions. An expression is defined in this context as a logical statement about the value(s) of a property of a concept, e.g. `amplifier:power = on`.[1] Examples of this type of relation are causal relations and time relations.

These four primitives are used to specify what we call a *domain schema* for a particular application. A domain schema is a description of the *structure* of the statements in the domain theory. It is roughly comparable to the notion of a signature in logic. For example, in a domain schema we could specify that the domain theory contains `part-of` relations between `component` concepts without worrying about the actual tuples of this relation. We prefer the term "schema" above " ontology" to stress the fact that the domain theory

[1] We use the shorthand <concept>:<property> for "the <property> of <concept>"

is the product of knowledge engineering and thus not necessarily describes an inherent structure in the domain (as the word "ontology" would suggest).

The domain schema specifies the main decisions that the knowledge engineer makes with respect to the *conceptualisation* [Genesereth & Nilsson, 1987; Nilsson, 1991] of the domain. For example, when a domain schema for a diagnostic domain is constructed, a decision has to be made whether "correct" or "fault" models (or both) are part of the domain theory. Parts of a domain schema often reappear in similar domains and could be reused (see Section 4 for a more detailed discussion of reusability). The domain schema also provides handles for describing the way in which inference knowledge uses the domain theory. Issues related to the interaction between domain knowledge and inference knowledge are discussed in the next section.

An example domain schema for a simple domain theory of diagnosing faults in a audio system is shown in Table 1.[2] Two types of concepts appear in this theory: *components* and *tests*. Both components and tests can have properties: respectively a *state-value* and a *value*. Two relations are defined between concepts of type "component": *is-a* and *part-of*. In addition, two relations between property expressions are defined: (i) a *causal* relation between state values of components, and (ii) an *indicates* relation between test values and state values.

Primitive	Name	Description
Concept	component	The elements of the audio system
Relation between concepts	component IS-A component	Sub-type hierarchy of components of the audio system
Relation between concepts	component SUB-COMPONENT-OF component	Part-of hierarchy of components of the audio system
Property	component:state-value	Components have properties describing the state that components are in at some moment in time.
Relation between expressions	component:state-value CAUSES component:state-value	Causal relations that specify how normal state-values of components are causally related to each other.
Concept	test	Test that can be performed to establish a state of an audio system.
Property	test:value	Possible outcomes of a test.
Relation between expressions	test:value INDICATES component:state-value	A relation describing which internal state is indicated by a particular test outcome.

Table 1: A domain schema for diagnosing faults in an audio-system

Figure 2 shows some domain knowledge in the audio domain. The domain knowledge description adheres to the structure defined in the domain schema of Table 1.

Domain knowledge can be viewed as a declarative theory of the domain. In fact, adding a simple deductive capability would enable a system in theory (but, given the limitations of theorem-proving techniques, not in practice) to solve all problems solvable

[2]The description of the domain schema given here is rather informal. For example, we do not say anything about cardinality (e.g. can a property have one or more values at some point in time). Techniques exist for describing these schemata in a more precise and formal way, e.g. [Davis & Bonnel, 1990; Hull & King, 1987]. Clearly, there is a close link in this area with the field of (hyper)-semantic database modelling.

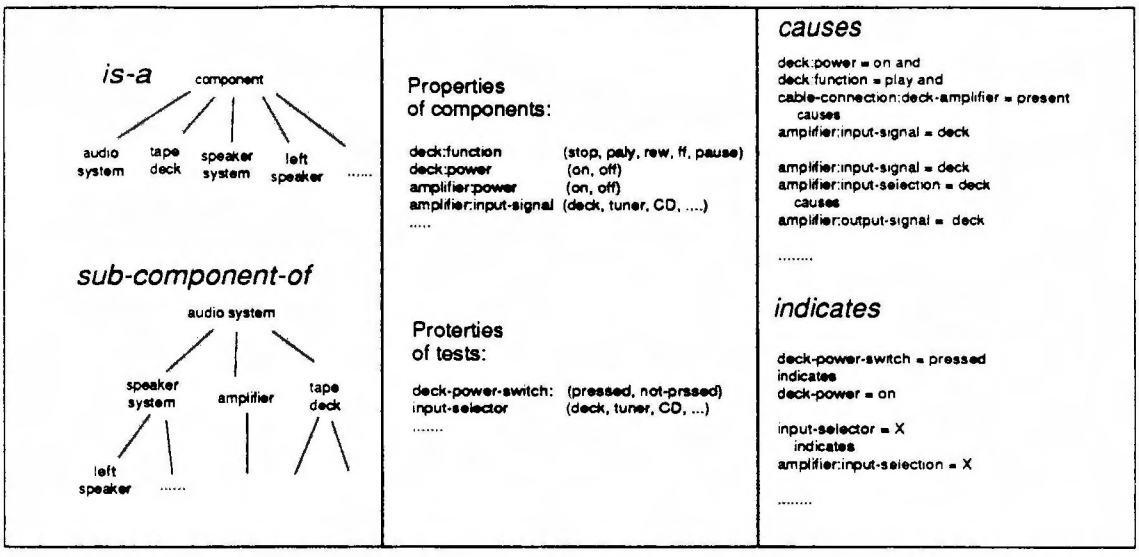


Figure 2: Domain knowledge of the audio system using the schema described in Table 1

by the theory. The domain knowledge is considered to be relatively task neutral, i.e. represented in a form that is independent of its use by particular problem solving actions. There is ample evidence [Wielinga & Bredeweg, 1988] that experts are able to use their domain knowledge in a variety of ways, e.g. for problem solving, explanation, teaching etc. Separating domain knowledge embodying the theory of the domain from its use in a problem solving process, is a first step towards flexible use and reusability of domain knowledge.

3.2 Inference Knowledge

At the first layer of control knowledge we abstract from the domain theory and describe the inferences that we want to make in this theory. We call this layer the *inference layer*. An inference specified at the inference level is assumed to be *primitive* in the sense that it it is fullly defined through its name, an input/output specification and a reference to the domain knowledge that it uses. The actual way in which the inference is carried out is assumed to be irrelevant for the purposes of modelling expertise. From the viewpoint of the model of expertise no control can be exercised on the internal behaviour of the inference. One could look upon the inference as applying a simple theorem prover. The primitive inferences are roughly comparable to the problem solving agents that Newell specifies at his knowledge level [Newell, 1982]. The behaviour of an inference is fully determined through a rationality principle such as Newell specifies:

> **Rationality principle:** if an agent has knowledge that one of its actions will lead to one of its goals, it will select this action.

Note that the inference is only assumed to be primitive with respect to the model of expertise. It is very well possible that such a primitive inference is realised in the actual system through a complex computational technique.

In the KADS model of expertise we use the following terms to denote the various aspects of a primitive inference:

Knowledge source

The action that is carried out in a primitive inference step is called a *knowledge source*. A knowledge source is an action that operates on some input data and has the capability of producing a new piece of information ("knowledge") as its output. During this process it uses domain knowledge. The name of the knowledge source is supposed to be indicative of the type of action that it carries out.

Meta class

A knowledge source operates on data elements and produces a new data element. We describe those data elements as *meta classes*. A meta class description serves a dual purpose:

(i) it acts as a *placeholder* for domain objects describing the *role* that these objects play in the problem solving process, and

(ii) it points to the *type(s)* of the domain objects that can play this role.

Domain objects can be linked to more that one meta class. For example, a particular component of an audio system could play the role of a *hypothesis* at one point and the role of *solution* at some other point in time. The name "meta class" is inspired by the fact that it provides a "meta" description of objects in a domain "class".[3] An input data element of a knowledge source is referred to as an *input* meta class; the output as an *output* meta class. Each meta class can be input and/or output of more than one knowledge source.

Domain view

The *domain view* specifies how particular parts of the domain theory can be used as domain knowledge by the knowledge source.

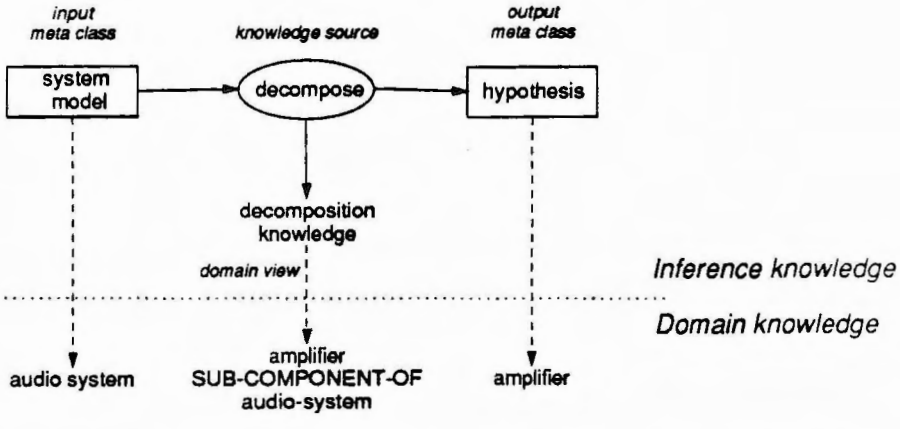


Figure 3: A primitive inference performing an decomposition action.

[3]It should not be confused with the meaning of this term in object-oriented systems.

Figure 3 describes a primitive inference in the audio domain with example references to domain knowledge. At the inference level a *decomposition* inference is specified. The action that is performed in this inference is the decomposition of a composite model of the audio system into sub-models. *System model* and *hypothesis* are examples of meta-classes. They describe the role that domain objects like `audio-system` and `amplifier` can play in the problem solving process. The *decompose* knowledge source achieves its goal, the generation of a new hypothesis, through the application of decomposition knowledge. The domain view of this inference specifies that tuples of the `SUB-COMPONENT-OF` relation in the domain theory can be used as decomposition knowledge. Figure 3 shows one applicable tuple of this relation.

A somewhat more formal specification of the *decompose* inference is given below. The arrow specifies how inference knowledge maps onto domain knowledge.

knowledge-source *decompose*
input-meta-class:
system-model → component
output-meta-class:
hypothesis → component
domain-view:
decomposition(system-model, hypothesis) → sub-component-of(component, component)

Note that this specification only refers to elements of the schema of the domain theory. Both *system model* and *hypothesis* are place holders of objects of type "component" and describe the role these objects play in the inference process. In this particular example the domain view refers to just one type of structure in the domain theory, namely the `SUB-COMPONENT-OF` relation. In principle however, there could be several of these mappings.[4]

There are distinct advantages of separating the domain theory from the way it is viewed and used by the inferences:

- The separation allows multiple use of essentially the same domain knowledge. Imagine for example a knowledge source *aggregate*, that takes as input a set of components and aggregates them into one composite component. This knowledge source could use the same `SUB-COMPONENT-OF` relation, but *view* it differently, namely as aggregation knowledge. Such an inference could very well occur in a system that performs configurations of audio systems.

- Domain knowledge that is used in more that one inference is specified only once. In this way, knowledge redundancy is prevented.

- It provides a dual way to *name*[5] domain knowledge: both use-independent and use-specific. Knowledge engineers tend to give domain knowledge elements names that already reflect their intended use and keep changing the names when their usage changes. We would argue that both types of names can be meaningful and should be known to the system, for example for explanation purposes.

[4] We omit here the details of specifying the mapping between a domain view and a domain theory. See for a more detailed discussion [Schreiber et al., 1989b; Chapter 3].

[5] We would argue that the whole activity of knowledge acquisition is in fact for a large part a matter of giving (meaningful) names.

- The scope of the domain theory is often broader than what is required for problem solving. For example, explanatory tasks (in KADS defined in the model of cooperation) often require deeper knowledge than is used during the reasoning process itself.

This is not to say that we claim that a domain theory can in general be defined completely independent of its use in the problem solving process. The scope and the structure of the domain knowledge has to meet the requirements posed by the total set of inferences. In many applications there are interactions between the process of conceptualising a domain and specifying the problem solving process. We still feel however, that it is useful to *document* them separately.

As stated previously, the primitive inference steps form the building blocks for an application problem solver. They define the basic inference actions that the system can perform and the roles the domain objects can play. The combined set of primitive inferences specifies the basic inference capability of the target system. The set of inference steps can be represented graphically in an *inference structure*. The inference structure thus specifies the problem solving *competence* of the target system.

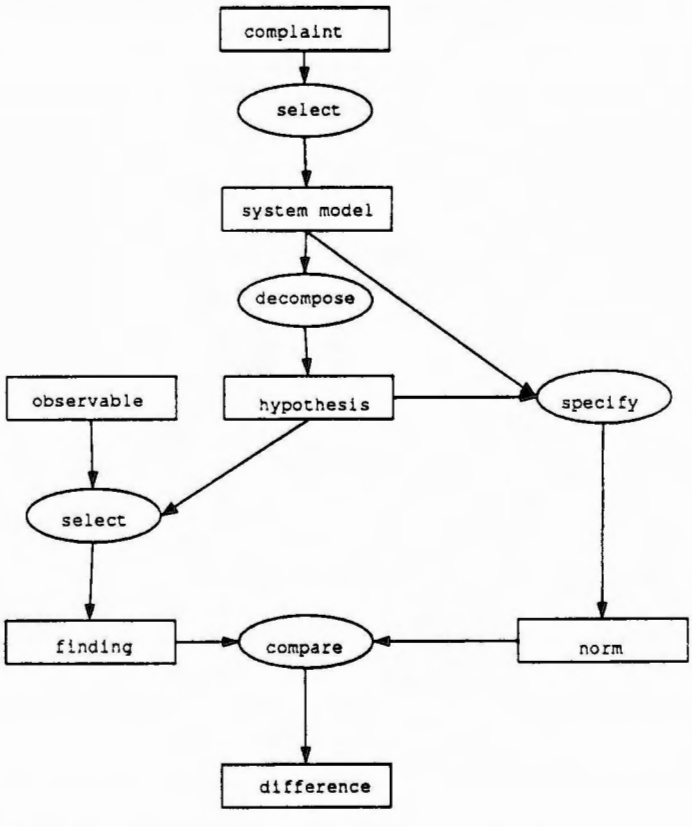


Figure 4: An inference structure for diagnosing faults in an audio system

Figure 4 presents such an inference structure for the audio domain. The inferences specify a top-down and systematic approach to find a sub-model of the audio system that behaves inconsistently. The following inferences appear in the inference structure:

- A (sub-part) of the audio system (*system model*) is selected on the basis of a *complaint.*
- This system model can be decomposed into a number of sub-components that play the role of *hypothesis.*
- For a hypothesis a *norm*-value can be predicted. The norm is a value of a test, that is consistent with the normal state of the hypothesis.
- This norm can be compared with the result of an observation (a *finding*). If these values differ, an inconsistency in the behaviour of the system has been found.

The inference structure defines the vocabulary and dependencies for control[6], but *not* the control itself. This latter type of knowledge is specified as task knowledge.

3.3 Task Knowledge

The third category contains knowledge about how elementary inferences can be combined to achieve a certain goal. The prime knowledge type in this category is the *task.* Tasks can achieve a particular *goal.* The relations between tasks and goals are in principle many-to-many. Task knowledge is usually characterised by a vocabulary of control terms, for instance indicating that a finding has been processed or a hypothesis has been verified.

Tasks represent fixed strategies for achieving problem solving goals. Several researchers [Clancey, 1985a; Gruber, 1989] have pointed out that task knowledge is an important element of expertise. The competence model of the diagnostic strategy of NEOMYCIN [Clancey, 1985a] is an example of what we call task knowledge. Clancey describes the sub-tasks of this strategy via meta-rules. The main difference approach between his approach and our approach is that he refers directly in these meta-rules to the domain knowledge. In KADS, tasks only refer to inferences and not explicitly to domain knowledge.

We use the following constructs to describe task knowledge:

Task

A task is a composite problem solving action. It implies a decomposition into sub-tasks. The result is a goal that can be achieved through the application of the task to a particular (sub-)problem.

Control terms

The control vocabulary used. For each control term a mapping is specified onto sets of meta class elements (e.g. the differential is the set of all active hypotheses).

Task structure

Specification of the decomposition into sub-tasks and their inter-dependencies.[7] The decomposition can involve three types of sub-tasks:

1. Primitive problem solving tasks: inferences specified in the inference layer.

[6] We use the term *control* here to refer to the process of controlling the execution of knowledge sources. We are not referring to more detailed, symbol-level forms of control such as search control in the application of a computational technique. See [Schreiber et al., 1990a] for a more elaborate discussion on these different types of control.

[7] We agree with [Steels, 1990] that "control structure" is a more appropriate term for this type of structure. We stick here to the term "task structure" mainly for historical reasons.

2. Composite problem solving tasks: a task specified in the task layer. In principle, this could be a recursive invocation of the same task.
3. Transfer tasks: tasks that require interaction with an external agent, usually the user.

The dependencies between the sub-tasks are described as a structured-English procedure with selection and iteration operators. The conditions in these procedures always refer to control terms and/or meta-class elements, e.g. "**if** *the differential is not empty* **then** ...".

Transfer tasks are more or less specified as a black box in the model of expertise. The transfer tasks are specified in the model of cooperation. Four types of transfer tasks are distinguished [de Greef et al., 1988a]:

1. *Obtain*: the system requests a piece of information from an external agent. The system has the initiative.
2. *Present*: the system presents a piece of information to an external agent. The system has the initiative.
3. *Receive*: the system gets a piece of information from an external agent. The external agent has the initiative.
4. *Provide*: the system provides an external agent with a piece of information. The external agent has the initiative.

An example task-knowledge specification for our audio domain is shown below. It consists of three tasks. The first task is *systematic-diagnosis*. The goal of this task is to find a sub-system with inconsistent behaviour at the lowest level of aggregation. The task works under the single-fault assumption. On the basis of a complaint, an applicable system model is selected. This selection task corresponds to the knowledge source *select* specified in the inference layer. Subsequently, hypotheses in the differential are generated through the *generate-hypotheses* sub-task. In the sub-task *test-hypotheses* these hypotheses are then tested to find an inconsistent sub-system. This hypothesis then becomes the focus for further exploration. The generate-and-test process is repeated, until no new hypotheses are generated (i.e. the differential is empty).

```
task systematic-diagnosis
  goal
      find the smallest component with inconsistent behaviour, if one.
  control-terms
      differential → set of currently active hypotheses
      inconsistent-sub-system → sub-part of the system with
        inconsistent behaviour
  task-structure
      systematic-diagnosis(complaint → inconsistent-sub-system) =
        select(complaint → system-model)
        generate-hypotheses(system-model → differential)
        repeat
```

```
    test-hypotheses(differential → inconsistent-sub-system)
    generate-hypotheses(inconsistent-sub-system → differential)
  until differential = ∅
```

For readability purposes, the names of knowledge sources are italicised in the task structure. The arrows in the task structure describe the relation between input and output of the sub-task. Note that all arguments of tasks and conditions are either explicitly declared control terms (*differential*) or meta-class names.

The task *generate-hypotheses* is a very simple task. It just executes the *decompose* knowledge source.

```
task generate-hypotheses
  goal
      generate new set of hypotheses through decomposition
  control-terms -
  task-structure
      generate(system-model → differential) =
        decompose(system-model → differential)
```

The task *test-hypotheses* tests the hypotheses in the differential sequentially until an inconsistency is found (*difference = true*). *Obtain(observable, finding)* is an example of a transfer task, that starts an interaction with the user to obtain a test value. How the transfer task is carried out, should be specified in the model of cooperation.

```
task test-hypotheses
  goal
      test whether a hypothesis in the differential behaves inconsistently
  control-terms -
  task-structure
      test(differential → hypothesis) =
        do for each hypothesis ∈ differential
          specify(hypothesis → norm)
          select(hypothesis → observable)
          obtain(observable → finding)
          compare(norm + finding → difference)
        until difference = true
```

If one abstracts from the control relations between sub-tasks, the set of task structures can be represented graphically as a tree. The tree for systematic diagnosis is shown in Figure 5.

3.4 Strategic Knowledge

The fourth category of knowledge is the *strategic knowledge*.[8] Strategic knowledge determines what goals are relevant to solve a particular problem. How each goal is achieved

[8] Gruber [Gruber, 1989] uses the term "strategic knowledge" in a different way. His strategic knowledge is in many aspects similar to the task knowledge in KADS.

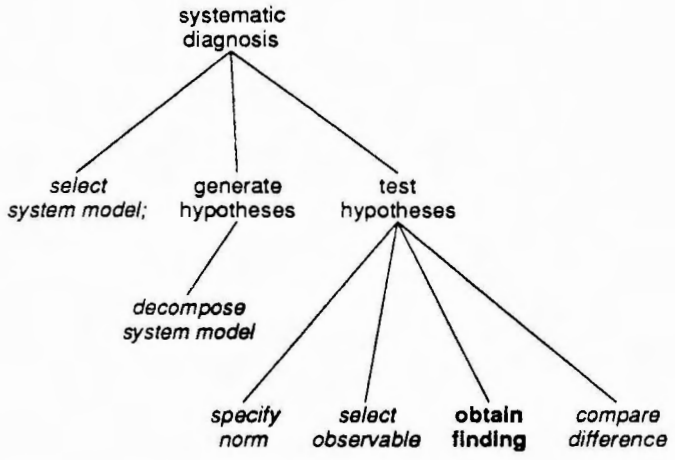


Figure 5: Task tree of systematic diagnosis. The leaves of such a tree are either knowledge sources or transfer tasks.

is determined by the task knowledge. Strategic knowledge will also have to deal with situations where the afore-mentioned knowledge categories fail to produce a partial solution. For example, the problem-solving process may reach an impasse because information is not available or because contradictory information arises. In such cases the strategic reasoning should suggest new lines of approach or attempt to introduce new information *e.g.*, through assumptions (cf. [Jansweijer, 1988; Jansweijer et al., 1989]).

Strategic knowledge concerns, among other things, the dynamic planning of task execution. However, most systems developed with the KADS approach used only fixed task decompositions and had little or no strategic knowledge. In our opinion, this does not mean that strategic knowledge is unimportant or superfluous. When knowledge engineers have to construct more complex and flexible knowledge-based systems than presently is usually the case, we think a much more detailed exploration of strategic knowledge will be necessary. We have recently started to work on an ESPRIT project named REFLECT where the central topic is the exploration of strategic knowledge. Apart from dynamic planning, strategic knowledge can also enable a system to answer questions such as "Can I solve this problem?" [Voß et al., 1990]. For the moment however, the study of the nature of strategic knowledge remains to be mainly a research topic.

3.5 Synopsis of the Model of Expertise

The four knowledge categories (domain, inference, task and strategic knowledge) can be viewed as four levels with meta-like relations in the sense that each successive level interprets the description at the lower level. In Figure 6 these four levels and their interrelations are summarised.

The four-level framework for knowledge modelling has been successfully used as a basis for structured acquisition and description of knowledge at an intermediate level between the verbal data obtained from experts and the knowledge representation in an implemented system [de Greef & Breuker, 1985]. From a knowledge-level viewpoint, the present four-layer model captures knowledge categories that are quite similar to those encountered in other models in the literature. However, differences in opinion exist about where to situate

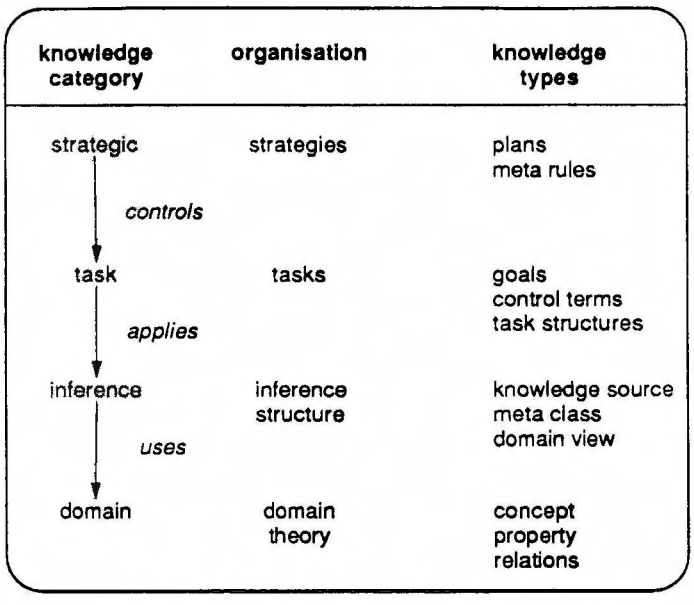


Figure 6: Synopsis of the KADS Four-Layer Model

particular types of knowledge. This point will be discussed in more detail in Section 6.

4 Reusable Model Parts

There are several ways in which models of expertise can be used to support the knowledge acquisition process. A potentially powerful approach is to *reuse* (structures of) model elements. When one models a particular application, it is usually already intuitively clear that large parts of the model are not specific for this application, but reoccur in other domains and/or tasks. KADS (as do most other approaches to knowledge modelling) makes use of this observation by providing a knowledge engineer with pre-defined sets of model elements. These libraries can be of great help to the knowledge engineer. They provide her with ready-made building blocks and prevent her from "re-inventing the wheel" each time a new system has to be built. In fact, we believe that these libraries are a *conditio sine qua non* for improving the state of the art in knowledge engineering.

In this section, the notion of an *interpretation model* is discussed as a way of reusing parts of the model of expertise.

4.1 Interpretation Models

A model of expertise from which the domain-specific knowledge layer is removed can be used by the knowledge engineer as a template for a new domain and thus support top-down knowledge acquisition. In KADS such partial models are called *interpretation models*, because they guide the interpretation of verbal data obtained from the expert.

The KADS interpretation models are models of expertise *with an empty domain layer*. Interpretation models describe typical *inference* and the *task* knowledge for a particular task. As these descriptions are phrased in domain-independent terminology, they are prime

candidates for reuse in other domains. For example, the inference and task description of the audio domain could very well be applicable to another domain where some device is being diagnosed. In [Breuker et al., 1987] interpretation models for a large number of tasks are presented. One of these is the model for systematic diagnosis as presented in this paper.

4.2 Example Interpretation Model

Another model in this library is that of the *monitoring* task. This model has been used in applications ranging from process control [Schrijnen & Wagenaar, 1988] to software project management [de Jong et al., 1988]. The inference structure (shown in Figure 7) describes the following inferences:

- The selection of a system parameter.
- The instantiation of the normal value of the parameter (the norm).
- The selection of a corresponding observable.
- A comparison of observed and expected values.
- A classification of the difference into a discrepancy class, e.g. *minor* or *major* disturbance. Often, data from previous monitoring cycles are used in this inference.

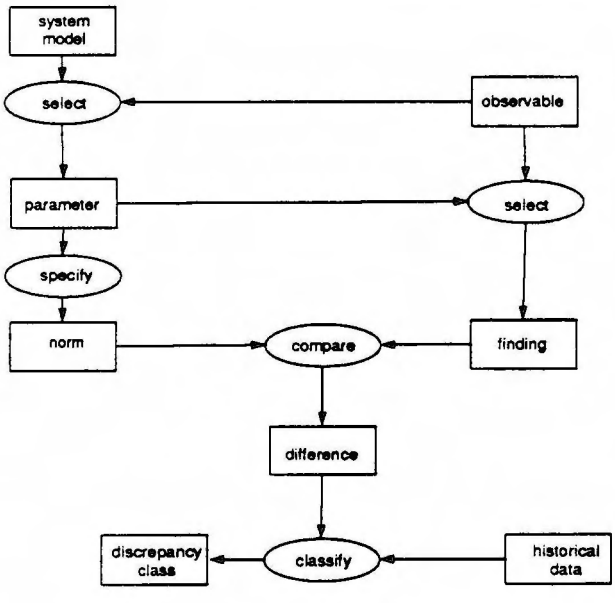


Figure 7: Inference structure of the interpretation model for *monitoring*

Two typical tasks (fixed strategies) were identified for monitoring. One could view them as two different ways of "going through" the inference structure of Figure 7.

The first task, *model driven monitoring*, describes a monitoring approach where the system has the initiative. This type of task is usually executed at regular points in time. The system actively acquires new data for some selected set of parameters and then checks whether the observed values differ from the expected ones.

```
task model-driven monitoring
  goal
      execute a monitoring cycle in which the system actively
      acquires new data
  control-terms
      active-parameters = set of parameter
  task-structure
      monitor(discrepancy) =
        select(system-model, active-parameters)
         do for each parameter ∈ active-parameters
           instantiate(parameter → norm)
           select(parameter → observable)
           obtain(observable → finding)
           compare(norm + finding → difference)
           classify(difference + historical-data → discrepancy)
```

The second task, *data-driven monitoring*, is initiated through incoming data. It contains a *receive* statement representing a transfer task (see Section 3.3) in which an external agent (a human user or another system) has the initiative. The values received are checked against expected values for the observables concerned. Resulting differences are subsequently classified in discrepancy classes.

```
task data-driven-monitoring
  goal
      execute a monitoring cycle when a new value of an observable
      is received by the system
  task-structure
      monitor(discrepancy) =
        receive(observable-set → finding)
         do for each observable ∈ observable-set
           select(observable + system-model → parameter)
           instantiate(parameter → norm)
           compare(norm + finding → difference)
           classify(difference + historical-data → discrepancy)
```

4.3 Selecting an Interpretation Model

The library of interpretation models consists of a number of models that could be used to describe the reasoning process in various applications. The knowledge engineer is guided in deciding which interpretation model to choose for a particular application through a decision tree. Part of this tree is shown in Figure 8. The decision tree is based on a taxonomy of task types. This taxonomy is a modified and extended version of Clancey's description of problem types [Clancey, 1985b]. The decision points in this tree concern features of the solution space, the problem space and the required knowledge types.

The first decision point concerns the structure of the system. The term "system" refers here to the central entity in the application domain, e.g. the audio system in the audio domain, the patient in a medical domain, the device in a technical domain, etc.

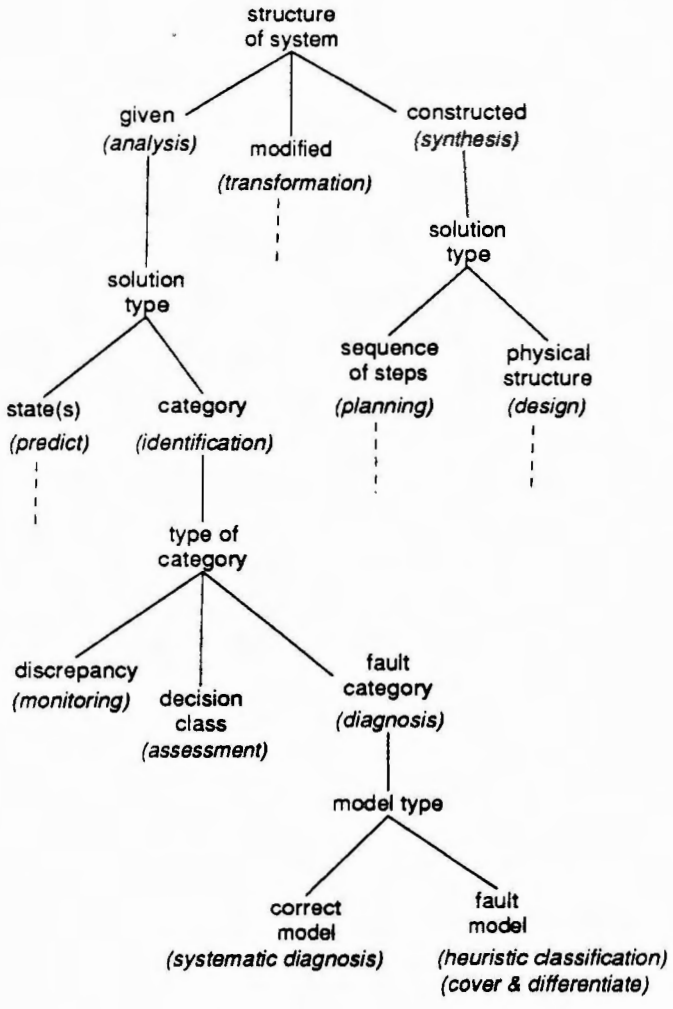


Figure 8: Partial decision tree of interpretation models

Other decision points concern for example the type of solution (state, category, types of categories, etc.) and the nature of the domain knowledge (fault-model or correct-model of the system). The leaves of the decision tree are associated with one or more interpretation models that specify typical inference and task knowledge for modelling this task. For example, the interpretation model for monitoring presented earlier is associated with the *monitoring* task in Figure 8. This model is chosen if (i) the structure of the system is given, (ii) the solution is a category and (iii) this solution category is not a fault category nor a decision class, but a simple discrepancy.

It should be noted that in many real-life applications there is more than one task in the task model for which an interpretation model needs to be chosen. For example, in the model of expertise for the audio domain, we focused only the diagnostic sub-task. In actual practice the repair/remedy task also needs to be addressed. This may result in a combination of (parts of) two or more interpretation models. An example of this process of combining is described in [Hayward, 1987].

5 Structure-Preserving Design

Unlike most other approaches to knowledge engineering, the KADS methodology as developed in the KADS-I project does not provide a dedicated computational framework that provides a straightforward operationalisation of models of expertise. This separation between modelling expertise on the one hand and a separate design step on the other hand has been identified as both the strength and the weakness of the KADS approach [Karbach et al., 1990].

The main advantage lies in the fact that the knowledge engineer is not biased during the process of modelling expertise by the restrictions of a computational framework. The four-layer model provides a more or less universal framework for modelling expertise and although computational constraints play a role in the construction of such models experience[9] has shown that this separation enables knowledge engineers to come up with more comprehensive specifications of the desired behaviour of the artefact.

The disadvantage lies in the fact that the knowledge engineer, after having built a model of expertise, is still faced with the problem of how to implement this specification. In this section we discuss some principles that can guide the knowledge engineer in this design process and also briefly discuss the issue of the computational adequacy of models of expertise.

It is important to realise that there is a trade-off between (i) extending the model of expertise or (ii) a more elaborate specification during design. The decision whether to do the first or the latter depends on whether one is interested from the model of expertise point of view in exercising explicit control on the execution of a computational technique. For example, in OFFICE-PLAN (a system for allocating offices to employees, [Karbach et al., 1989]) the actual allocation inference is modelled as one knowledge source *assign* in the model of expertise and is realised in the actual system through a complex constraint satisfaction technique. If it would have been necessary to exercise control on this technique, then one would need to model constraint satisfaction "at the knowledge level".

5.1 Design Decisions

By definition, knowledge-level models such as the KADS models of expertise do not contain all information necessary for the implementation of a system. The design problem can be defined as the selection (or possibly development) of appropriate representations and computational techniques for the elements in this model.

In principle, the designer is free to make any set of design decisions that results in meeting the specifications. In fact, a number of successful applications have been built that used standard design techniques [Readdie & Innes, 1987a; Benus & van der Spek, 1988]. Often this was due to external requirements that forced the use of, for example, first generation expert system shells that supply only a very limited set of AI techniques.

From a methodological point we favour however strongly a *structure preserving design*. With structural preserving we mean that there should be a correspondence between model of expertise elements and computational constructs. Design is viewed as a process of *adding* symbol-level information to a model of expertise.

[9] See Section 8 for an overview of applications developed with the KADS approach

Generally, the following design decisions have to be made with respect to the elements of the model of expertise [Schreiber et al., 1987; Schreiber et al., 1989a]:

Inference knowledge For each knowledge source a corresponding computational technique needs to be selected that can realise this inference. A technique consists of three types of elements: (i) an algorithm, (ii) input-output data structures and possibly additional temporary data structures, and (iii) a representation of domain knowledge.

The algorithm embodies the method for realising the inference and specifies the local, symbol-level, control. The representation of the input-output data structures corresponds to the meta-classes and should be synchronised within the set of knowledge sources. The chosen domain knowledge representation (e.g. production rules) implies the requirement that the representation of the domain theory can be *viewed* in this way.

In AI research a number of (groups of) computational techniques have been developed such as production systems, state-space search, parsing, classification and matching. Detailed studies have been performed to unravel the criteria for choosing a technique within one group such as hierarchical classification [Goel et al., 1987] and logical representation and deduction [Reichgelt & van Harmelen, 1986]. In [Schreiber et al., 1989a] criteria for chosing a particular type of technique are discussed. Often, knowledge engineers use within one system only one or two types of techniques. In that case the chosen type of technique takes the role of a design *paradigm*. For example, in NEOMYCIN [Clancey, 1985a] four (forward chaining) production system techniques are provided. Each inference applies one of these techniques. Applying only one type of technique in a system minimises the interaction problems with the design of other parts of the system (in particular the domain theory, see further), but apart from that there is no compelling reason to adhere strictly to this approach.

Domain knowledge We view the domain knowledge as an elaborate database (with much more sophisticated representational primitives than conventional databases apply) and the computational techniques as requiring an access path to this database. A crucial design decision concerns the choice of the representation technique(s) for the domain theory. This representation has to meet the requirement posed by the set of "domain views". In other words, one should be able to view (e.g. rewrite) the domain representation in such a way that it meets the demands of the computational techniques implementing knowledge sources. In addition, the representational technique should allow the specification of knowledge needed for other purposes, such as explanation. These requirements usually follow from the specification of the model of cooperation.

If the knowledge engineer works within a particular paradigm, such as production systems, the choice of the representational technique is usually obvious: the domain representation technique is the same as the representation used by the chosen type of computational technique.

In most existing systems no clear difference is made between use-specific and use-independent names of elements of the domain-theory. This violates however the structural correspondence requirement and thus implies that the advantages of separating these two (see Section 3.2) do not hold for the final system. A violation can create problems, particularly in the area of maintenance. For example, it could lead to multiple presence in the domain theory of essentially the same knowledge.

Task knowledge Given the set of tasks specified in the model of expertise (consisting of both problem solving tasks and of transfer tasks) the designer basically has to make two – inter-related – decisions, namely:

1. The choice of a control technique for executing tasks. Examples of such control techniques are an agenda mechanism, a blackboard, a skeletal planning technique or a simple procedure hierarchy.
2. The choice of how to represent and update working memory. With "working memory" we mean here the data that are manipulated by the tasks during the problem solving process. The control terms and the meta classes specified in the model of expertise should reappear as labels of (sets of) working memory elements. Most existing KBS's use a simple monotonic technique of updating working memory. We expect that in the next generation KBS's more complex techniques such as truth-maintenance will be used more often. Note that the use of such a technique can pose additional requirements on the output produced by computational techniques realising primitive inferences. An example of such an additionally required output is the "justification" used in an ATMS [de Kleer, 1986].

Strategic knowledge Most models of expertise that have been constructed do not contain a large amount of strategic knowledge. In many of these systems the strategic part has been "compiled out" through fixed task decompositions with possibly a few strategic decision points that can be influenced by the user (cf. [de Greef et al., 1987; Part II, Section 3.4]).

If more elaborate strategic knowledge is present, the following techniques could be applicable:

- A production system containing a set of meta-rules with as conditions states of working memory and as actions task activations and/or changes to working memory (e.g. assumptions).
- An extended blackboard technique such as the "Blackboard Control Architecture" [Hayes-Roth, 1985], where the scheduling part represents the strategic knowledge.
- One can also view the strategic knowledge as a separate meta-system, This approach is currently being pursued in the REFLECT project [Bartsch-Sporl et al., 1990].

The design decisions should thus lead to a clear mapping from the elements of the model of expertise onto elements of the implementation environment. Such a structure-preserving design has a number of advantages:

Explanation

One important advantage is a wider possible scope for explanations that can be generated by the system. First generation expert systems were only able to give explanations by paraphrasing their code (e.g. MYCIN). It is now commonly accepted that this is not sufficient to understand the reasoning process of the KBS. Preserving the structure of the model of expertise makes it possible to generate explanations at the level of the language of the model of expertise (cf. [Clancey & Letsinger, 1984]).

Maintenance and debugging

An other advantage has to do with the maintenance and the debugging of a KBS. The preservation of the structure of the model of expertise makes it possible to trace an omission or inconsistency in the implemented artefact back to a particular part of the model of expertise. Also, knowledge redundancy is prevented.[10]

Knowledge acquisition tools

A related advantage is that the model of expertise and its links to the artefact can be used to guide the use of knowledge acquisition tools and techniques. For example, techniques such as repertory grid and induction algorithms can be used to generate not *any* knowledge but a *particular type* of knowledge. We think that this approach to knowledge acquisition support allows much more focused use of tools and therefore increases the interpretability and the quality of their results. The approach sketched here is currently being pursued in the ACKNOWLEDGE project [Shadbolt & Wielinga, 1990].

In KADS we have developed a notation for supporting this structure preserving design process through intermediate design descriptions [Schreiber et al., 1989b]. To support the knowledge engineer in this transformation approach it is also possible to develop a dedicated computational framework that provides:

- a number of pre-defined representational and computational techniques, and
- a programming environment that more or less pre-defines how elements of the model of expertise should be mapped onto constructs in this environment.

In such a computational framework most of the design decisions discussed earlier are hard-wired. The use of a dedicated framework enables the knowledge engineer to go straight from a model of expertise to an implementation. Model-K [Karbach et al., 1991] and ZDEST-2 [Tong et al., 1988; Karbach et al., 1988] are computational frameworks that allow (partially) such a mapping of KADS models of expertise onto an implementation. A potential problem with this approach is that the framework may not provide all the necessary techniques for a particular application. As any other design process, the design of a knowledge-based system is by its nature open-ended: i.e. the solution space is infinite.

5.2 Computational Adequacy

One major objection to the use of knowledge-level models in the KBS development process is the potential computational inadequacy of such models. Since knowledge-level models do not specify the control regime in full detail they are apt to potential combinatorial explosion behaviour. Although it is true in principle that this problem may occur, the structure of the models proposed here provides important safeguards against the computational inadequacy. The knowledge that is specified in a KADS four-layer model cannot be used in arbitrary ways, it has to fulfill certain typing requirements and can only be

[10] Similar goals with respect to explanation and maintenance are pursued in the Explainable Expert Systems (EES) approach [Neches et al., 1985]. In EES the structure preserving property is guaranteed through an automatic transformation process.

applied within the constraints specified by the model. For instance, a logical implication of a certain type can only be used for abstraction inferences, not for all types of inference steps. Our introduction of knowledge sources and task structures generally yields the possibility of selecting specific rules or theories needed to produce a certain inference. In this way the knowledge-level model provides a *role-limiting* [McDermott, 1989] constraint to knowledge. Role-limitations translate to *access* limitations at the symbol level and hence reduce the combinatorial explosion.

6 Relation to Other Approaches

We make no claim that all ideas underlying KADS are new. On the contrary, work of other researchers has heavily influenced the work on KADS. In this section we discuss a number of these approaches and relate them to the KADS approach.

Brachman [Brachman, 1979] proposed five levels for describing knowledge: the linguistic, the conceptual, the epistemological, the logical and the implementational level. Work of Clancey [Clancey, 1983] showed that the epistemological level of Brachman is the "missing" level in the description of knowledge-based systems. We interpret Newell's knowledge level as a combined description of the conceptual and epistemological level. In the KADS model of expertise the domain knowledge roughly corresponds to the conceptual level and the three other categories to the epistemological level. The KADS design description (Newell's symbol level) corresponds to the logical level.

The work of Clancey has had a large impact on KADS. In [Clancey, 1985b] he introduced the notion of an inference structure in the description of the model of heuristic classification (HC). (the "horse-shoe" diagram). In the work on NEOMYCIN [Clancey, 1985a] the same type of task decomposition is found as appears in the task layer in KADS. The main difference is that there is no explicit relation between the tasks in NEOMYCIN and the inferences in the HC model. These tasks refer directly to the domain knowledge, whereas the tasks in KADS reference the domain knowledge only indirectly via primitive inferences.

A number of approaches do not make the distinction between knowledge-level modelling and design as favoured by KADS. In these approaches the starting point is an implementation environment. A knowledge-level model is used as an implementation-independent description of constructs of this environment.

In the approach taken at Ohio State University [Bylander & Chandrasekaran, 1988; Chandrasekaran, 1988] the implementation environment consists of so called "generic tasks". A generic task (GT) is a combination of a problem (e.g. classification) and a problem solving method (e.g. hierarchical classification) with particular knowledge and inference requirements. GT's can perform quite general information-processing tasks. The assumption is that by combining a relatively small set of GT's one can solve a large number of problems. The problem solving methods in the GT approach have a somewhat smaller grain size than the interpretation models in KADS.

In the approach taken at DEC [McDermott, 1989; Marcus & McDermott, 1989; Eshelman et al., 1988] a number of systems where built that provided an operationalisation of a particular problem solving method, such as "propose & revise" and "cover & differentiate". The terminology used to describe these methods is such that during knowledge acquisition the expert can be prompted for domain knowledge in a high-level, method-specific,

language, e.g. "`What are symptoms that the system should be able to explain?`". The problem solving methods have a similar grain size as the KADS interpretation models. More recently [Klinker et al., 1990], the emphasis in this approach has shifted to the construction of an integrated environment in which the knowledge engineer can *configure* such single-task knowledge acquisition systems from a set of pre-defined mechanisms. The research on a typology of mechanisms is very close to aims in KADS.

In the PROTEGE [Musen, 1989] approach the problem is addressed that experts find it difficult to enter knowledge in a method-specific format. They distinguish two steps in building ONCOCIN-like systems: the knowledge engineer uses PROTEGE to specify the required domain knowledge in method-specific terms; PROTEGE then generates a knowledge acquisition tool called p-OPAL that enables the expert to enter knowledge in domain-specific terms. This dual way of naming domain knowledge is similar to the approach advocated in KADS. The PROTEGE system presupposes a single-task model, based on the skeletal planning method of ONCOCIN [Shortliffe et al., 1981].

All models used in these last three approaches are hard-wired to particular computational constructs. As stated earlier, compared to the KADS approach this is both an advantage and a disadvantage.

The "Components of Expertise" (CoE) approach [Steels, 1990] is in many aspects similar to KADS. The main differences compared to KADS are the dynamic view on task decomposition based on task features and the absence of an explicit description of inference knowledge such as meta classes. A dedicated computational framework has been developed for CoE models [Vanwelkenhuysen & Rademakers, 1990]. Research aiming at a synthesis of KADS and CoE is in progress within the KADS-II project.

The "Ontological Analysis" approach [Alexander et al., 1988] describes knowledge in three categories: (i) the static ontology describing the primitive objects, properties and relations, (ii) the dynamic ontology describing the state space of the problem solver and the actions that can make transitions in this space, and (iii) the epistemic ontology describing methods that control the use of knowledge of the first two categories. These three categories resemble closely the domain, inference and task knowledge in KADS. The formalisms used in Ontological Analysis are based on algebraic specification languages.

Although terminology is different, a common view appears to be emerging based on the idea that different types of knowledge constitute the knowledge level and that these different types of knowledge play different roles in the reasoning process and have inherently different structuring principles. One salient characteristic is that all approaches distinguish between structural domain knowledge and control knowledge. In addition, various kinds of control knowledge are distinguished, like global control of how to go about the task as a whole, and local control knowledge specifying how and/or when to carry out certain individual actions.

7 Experiences

The KADS approach has been (and is being) used in some 40 to 50 KBS projects. Not all these projects used "pure" KADS. The core activities of Bolesian Systems, a Dutch company, are teaching and applying an earlier version of KADS under the name SKE (Structured Knowledge Engineering). Other companies, such as Arthur Andersen Consulting and commercial partners in the KADS-I project, have incorporated KADS into their

own methodology.

Within the KADS-I project the approach has been tested in a number of experiments in domains such as commercial wine making [Wielinga & Breuker, 1984] statistical consultancy [de Greef & Breuker, 1985; de Greef et al., 1988b], network management [Krickhahn et al., 1988; Readdie & Innes, 1987b], mould configuration [Barthélemy et al., 1988], mixer configuration [Wielemaker & Billault, 1988], technical diagnosis [Wright et al., 1988] and and credit card fraud detection [Land et al., 1990]. Other applications include process control [Schrijnen & Wagenaar, 1988], chemical equipment [Schachter & Wermser, 1988], room planning [Karbach et al., 1989], social security [de Hoog, 1989], software project management [de Jong et al., 1988], diagnosis of movement disorders [Winkels & Achthoven, 1989], and paint selection [van der Spek et al., 1990]. The last two systems and the credit card system are already being used for some time in practice. A recent publication for the commercial AI community [Harmon, 1991] commented that "before KADS, most of the methodologies were vague prescriptions rather than systematic step-by-step models for large scale systems development efforts". On the basis of the success of KADS-I, the CEC has decided to fund a second ESPRIT project (KADS-II) with the aim to arrive at a *de facto* European standard for KBS development.

This is not to say that we think that the approach described in this paper has no deficiencies: on the contrary. It is clear that a group of KADS users finds certain aspects of KADS attractive, but it is being recognised that there are many weaknesses in its current state. The first KADS user meeting [Ueberreiter & Voß, 1991] in which some forty, mainly German, people participated, provided a good overview of the the strong and weak points of KADS. Among the strong points are:

- The distinction between various models, especially the distinction between the model of expertise and the design.
- The framework for modelling expertise. Especially the inference structures are mentioned by many people as an intuitively appealing way of describing the reasoning process and as a communication vehicle with domain experts.
- The library of interpretation models. Although this library is far from complete, it has still provided useful starting points for many applications.

The list of weaknesses is considerably longer. A selection:

- The vocabulary in the four-layer framework for describing domain knowledge and task knowledge is not expressive enough.
- The typology of knowledge sources is too general. The precise meaning of the knowledge sources is ambiguous.
- The library of interpretation models is incomplete and needs serious revision. For example, coverage of synthetic tasks is marginal.
- KADS does not provide enough support for operationalising models of expertise.
- KADS gives you a vocabulary, but it provides little support for the actual knowledge engineering process.

In short, the experiences show that the KADS has some interesting and attractive features, but that it still needs a lot of work before it can really be considered to be

"comprehensive methodology". In addition, controlled validation studies are necessary to show that KADS actually provides advantages when compared to other approaches.

8 Future Developments & Conclusions

In this paper we have taken the position that knowledge acquisition is to a large extent a constructive activity: models of several aspects of the task and domain have to be build before implementing a knowledge based system.

Looking at the future of knowledge acquisition from this point of view, raises the obvious question of how AI and knowledge based systems themselves can support the various modelling processes. Recent developments in the area of knowledge acquisition tools provide some directions in how this could be done.

Given the modelling approach to knowledge acquisition it is of vital importance that a knowledge engineer has some language in which the various models can be formulated. Such a language is not only important for the knowledge acquisition process itself, but also for communicating models and comparing models for different tasks. A comparative analysis of models of human expertise and of expertise embodied in KBS's will advance the knowledge acquisition activity from an art to a proper engineering discipline. Although there is currently little consensus on what the ingredients and vocabulary of such a modelling language should be, the various ideas appear to converge. The result of the synthesis of the KADS and the CoE approach, which is currently being pursued and in which ideas from other approaches are also taken into account, may be a starting point for such a language. In our view, it is worthwhile to investigate the different types of knowledge and their relationships from a more formal point of view. A first attempt has been made in this direction [Akkermans et al., 1990; Schreiber et al., 1990b]. Such a formal account of knowledge models clarifies at least some of the notions that have been used in a rather informal way so far.

If a common language for defining conceptual models of problem solving processes became accepted, it would be of great interest to study the large collection of problem solving models that currently exist. A consolidation and integration of the models in the KADS interpretation model library [Breuker et al., 1987], the generic problem solving models of Chandrasekaran and co-workers [Chandrasekaran, 1988], the models underlying the various model-driven knowledge acquisition tools [McDermott, 1989], and various other models in the literature, could provide the knowledge engineering community with a invaluable tool for knowledge acquisition. Also, such a collection of generic models could be the basis of a powerful knowledge acquisition tool that could both communicate with experts and with knowledge engineers.

Looking beyond the traditional knowledge engineering paradigm where the knowledge engineer does most of the work, we envisage an important role for knowledge about models in knowledge acquisition tools that integrate traditional knowledge acquisition techniques and automated learning techniques. One of the major problems in this area is that of integrating knowledge of various sources. A system that has knowledge about the kinds of knowledge that it needs to acquire can exercise much more focused control on the acquisition process and hence solve at least part of the integration problems.

Acknowledgements

This research was largely carried out in the course of the KADS-I project. This project was partially funded by the ESPRIT-I Programme of the Commission of the European Communities as project number 1098. The partners in this project were STC Technology Ltd., SD-Scicon plc., KBSC of TRMC (all United Kingdom), NTE NeuTech (Germany), Cap Sesa Innovation (France), and the University of Amsterdam (The Netherlands). We gratefully acknowledge the contributions to the work reported here from Anjo Anjewierden, Jean-Paul Billault, Bert Bredeweg, Massoud Davoodi, Paul de Greef, Simon Hayward, Robert de Hoog, Ton de Jong, Maarten van Someren, Peter Terpstra and Jan Wielemaker. Many other co-workers of ESPRIT Project 1098 participated in discussions, applications and validations of the ideas presented here. Anjo Anjewierden, Frank van Harmelen and Jacobijn Sandberg provided extensive comments on earlier versions of this paper.

References

Akkermans, H., van Harmelen, F., Schreiber, G., & Wielinga, B. (1990). A formalisation of knowledge-level models for knowledge acquistion. *International Journal of Intelligent Systems*. Forthcoming.

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

Barthélemy, S., Frot, P., & Simonin, N. (1988). Aanalysis document experiment f4. ESPRIT Project P1098, Deliverable E4.1, Cap Sogeti Innovation.

Bartsch-Sporl, B., Reinders, M., Akkermans, H., Bredeweg, B., Christaller, T., Drouven, U., van Harmelen, F., Karbach, W., Schreiber, G., Voß, A., & Wielinga, B. (1990). A tentative framework for knowledge-level reflection. ESPRIT Basic Research Action P3178 REFLECT, Deliverable IR.2 RFL/BSR-ECN/I.3/1, BSR Consulting and Netherlands Energy Research Foundation ECN.

Benus, B. & van der Spek, R. (1988). The Paint Expert: Report on the development of a knowledge-based system for naive users. Master's thesis, University of Amsterdam, Faculty of Psychology.

Brachman, R. (1979). On the epistemological status of semantic networks. In Findler, N., editor, *Associative Networks*, New York. Academic Press.

Breuker, J. & Wielinga, B. (1987). Use of models in the interpretation of verbal data. In Kidd, A., editor, *Knowledge Acquisition for Expert Systems, a practical handbook*, New York. Plenum Press.

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

Breuker, J. A. & Wielinga, B. J. (1989). Model Driven Knowledge Acquisition. In Guida, P. & Tasso, G., editors, *Topics in the Design of Expert Systems*, pages 265–296, Amsterdam. North Holland.

Bylander, T. & Chandrasekaran, B. (1988). Generic tasks in knowledge-based reasoning: The 'right' level of abstraction for knowledge acquisition. In Gaines, B. & J.Boose, editors, *Knowledge*

Acquisition for Knowledge Based Systems, volume 1, pages 65–77. Academic Press, London.

Chandrasekaran, B. (1988). Generic tasks as building blocks for knowledge-based systems: The diagnosis and routine design examples. *The Knowledge Engineering Review*, 3(3):183–210.

Clancey, W. (1983). The epistemology of a rule based system -a framework for explanation. *Artificial Intelligence*, 20:215–251. Also: Stanford Heuristic Programming Project, Memo HPP-81-17, November 1981, also numbered STAN-CS-81-896.

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

Clancey, W. (1985b). Heuristic classification. *Artificial Intelligence*, 27:289–350.

Clancey, W. & Letsinger, R. (1984). NEOMYCIN: Reconfiguring a rulebased expert system for application to teaching. In Clancey, W. & Shortliffe, E., editors, *Readings in Medical Artificial Intelligence: the first decade*, pages 361–381. Addison-Wesley, Reading.

Davis, J. & Bonnel, R. (1990). Producing visually-based knowledge specifications for acquiring organizational knowledge. In Wielinga, B., Boose, J., Gaines, B., Schreiber, A., & van Someren, M., editors, *Current Trends in Knowledge Acquisition*, pages 105–122, Amsterdam. IOS Press.

Davis, R. (1980). Metarules: Reasoning about control. *Artificial Intelligence*, 15:179–222.

de Greef, P. & Breuker, J. (1985). A case study in structured knowledge acquisition. In *Proceedings of the 9th IJCAI*, pages 390–392, Los Angeles.

de Greef, P. & Breuker, J. (1989). A methodology for analysing modalities of system/user cooperation for KBS. In Boose, J., Gaines, B., & Ganascia, J., editors, *Proceedings European Knowledge Acquisiton Workshop, EKAW'89*, pages 462–473, Paris, France. AFCET/ARC/ECCAI.

de Greef, P., Breuker, J., & de Jong, T. (1988a). Modality: An analysis of functions, user control and communication in knowledge-based systems. Esprit Project P1098, Deliverable D6 (task A4), University of Amsterdam.

de Greef, P., Breuker, J., Schreiber, G., & Wielemaker, J. (1988b). StatCons: Knowledge acquisition in a complex domain. In *Proceedings ECAI'88*, Munich.

de Greef, P., Schreiber, G., & Wielemaker, J. (1987). The StatCons case study. ESPRIT Project P1098, Deliverable E2.3 (experiment F2), University of Amsterdam.

de Hoog, R. (1989). Een expertsysteem, bijstand voor bijstand. *Informatie & Informatiebeleid*, 7(1):47–53.

de Jong, T., de Hoog, R., & Schreiber, G. (1988). Knowledge acquisition for an integrated project mnagament system. *Information Processing and Management*, 24(6):681–691.

de Kleer, J. (1986). An assumption-based TMS. *Artificial Intelligence*, 28:127–162.

DeMarco, T. (1982). *Controlling Software Projects.* Yourdon Press, New York.

Eshelman, L., Ehret, D., McDermott, J., & Tan, M. (1988). MOLE: a tenacious knowledge acquisition tool. In Boose, J. & Gaines, B., editors, *Knowledge Based Systems, Volume 2: Knowledge Acquisition Tools for Expert Systems*, pages 95–108, London. Academic Press.

Genesereth, M. & Nilsson, N. (1987). *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann, Los Altos, California.

Goel, A., Soundarajan, N., & Chandrasekaran, B. (1987). Complexity in classificatory reasoning. In *AAAI-87*, pages 421–425.

Gruber, T. (1989). *The Acquisition of Strategic Knowledge*. Perspectives in Artificial Intelligence, Volume 4. Academic Press, San Diego.

Harmon, P. (1991). A brief overview of software methodologies. *Intelligent Software Strategies*, VII(1):1–19. Newsletter. Circulation office: 37 Broadway, Arlington. MA 02174 USA.

Hayes-Roth, B. (1985). A blackboard architecture for control. *Artificial Intelligence*, 26(3):251–321.

Hayward, S. (1987). How to build knowledge systems; techniques, tools, and case studies. In *Proceedings of 4th annual Esprit conference*, pages 665–680, Amsterdam. North-Holland.

Hayward, S., Wielinga, B., & Breuker, J. (1987). Structured analysis of knowledge. *International Journal of Man-Machine Studies*, 26:487–498.

Hull, R. & King, R. (1987). Semantic dtabase modelling: Survey, applications. and research issues. *ACM Computing Surveys*, 19:201–260.

Jansweijer, W. (1988). *PDP*. PhD thesis, University of Amsterdam.

Jansweijer, W., Elshout, J., & Wielinga, B. (1989). On the multiplicity of learning to solve problems. In Mandl, H., de Corte, E., Bennett, N., & Friedrich, H., editors, *Learning and Instruction: European research in an international context*, pages 127–145. Pergamon Press, Oxford, UK.

Karbach, W., Linster, M., & Voß, A. (1989). OFFICE-PLAN: Tackling the synthesis frontier. In Metzing, D., editor, *GWAI-89: 13th German Workshop on Artificial Intelligence. Informatik Fachberichte 216*, pages 379–387, Berlin. Springer Verlag.

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

Karbach, W., Tong, X., & Voß, A. (1988). Fillig in the knowledge acquisition gap: via KADS models of expertise to ZDEST-2-expert systems. In *Proceedings of EKAW '88*, Bonn.

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

Klinker, G., Bhola, C., Dallemagne, G., Marques, D., & McDermott, J. (1990). Usable and reusable programming constructs. In *Proceedings 5th AAAI-sponsored Knowledge Acquisition for Knowledge Based Systems Worksho, Banff, Canada*, pages 14–1 – 14–20, Alberta, Canada. SRDG Publications, University of Calgary.

Krickhahn, R., Nobis, R., Mahlmann, A., & Schachter, M. (1988). Applying the kads methodology to develop a knowledge-based system. In *Proccedings ECAI'88, Munich*, pages 11–17. London. Pitman.

Land, L., Taylor, R., Mulhall, T., Killin, J., & Bingham, T. (1990). Credit card fraud identification knowledge-based system. ESPRIT Project P1098, deliverable F12.1, KADS-F12-D1-001. KBSC of Touche Ross Management Consultants, London.

Marcus, S. & McDermott, J. (1989). SALT: A knowledge acquisition language for propose-and-revise systems. *Artificial Intelligence*, 39(1):1–38.

McDermott, J. (1989). Preliminary steps towards a taxonomy of problem-solving methods. In Marcus, S., editor, *Automating Knowledge Acquisition for Expert Systems*, pages 225–255. Kluwer Academic Publishers, The Netherlands.

Meyer, M. & Booker, J. (1991). *Eliciting and Analyzing Expert Judgement: A Practical Guide*,

volume 5 of *Knowledge-Vase Systems*. Academic Press, London.

Morik, K. (1989). Sloppy modelling. In Morik, K., editor, *Knowledge Representation and Organisation in Machine Learning*. Springer Verlag.

Musen, M. (1989). *Automated Generation of Model-Based Knowledge-Acquisition Tools*. Pitman, London. Research Notes in Artificial Intelligence.

Neale, I. (1988). First generation expert systems: a review of knowledge acquisition methodologies. *The Knowledge Engineering Review*, (2):105–145.

Neches, R., Swartout, W., & Moore, J. (1985). Enhanced maintenance and explanation of expert systems through explicit models of their development. *IEEE Trans. Softw. Eng.*, 11:1337–1351.

Newell, A. (1982). The knowledge level. *Artificial Intelligence*, 1982:87–127.

Nilsson, N. (1991). Logic and artificial intelligence. *Artificial Intelligence*, 47:31–56.

Pople, H. (1982). Heuristic methids for imposing structure on ill-structured problems: The structuring in medical diagnosis. In Szolovits, P., editor, *Artificial Intelligence in Medicine*, pages 119–190. Westview Press, Boulder CO.

Readdie, M. & Innes, N. (1987a). Network management: Kbs design document. ESPRIT Project P1098, Deliverable E3.2a, SciCon Ltd. (UK).

Readdie, M. & Innes, N. (1987b). Network management: Requirements analysis and feasibility analysis. ESPRIT Project P1098, Deliverable E3.1a, SciCon Ltd. (UK).

Reichgelt, H. & van Harmelen, F. (1986). Criteria for choosing represenation languages and control regimes for expert systems. *Knowledge Engineering Review*, 1:2–17.

Schachter, M. & Wermser, D. (1988). A sales assistant for chemical measurement equipment. In *Proccedings ECAI'88, Munich*, pages 191–193, London. Pitman.

Schreiber, G., Akkermans, H., & Wielinga, B. (1990a). On problems with the knowledge level perspective. In Boose, J. & Gaines, B., editors, *Proceedings Banff'90 Knowledge Acquisition Workshop*, pages 30–1 – 30–14, University of Calgary. SRDG Publications. Also in: Proc. AISB, 1991.

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

Schreiber, G., Bredeweg, B., de Greef, P., Terpstra, P., Wielinga, B., Brumet, E., Simonin, N., & Wallyn, A. (1989a). A KADS approach to KBS design. ESPRIT Project 1098, deliverable B6 UvA-B6-PR-010, University of Amsterdam & Cap Sogeti Innovation.

Schreiber, G., Breuker, J., Bredeweg, B., & Wielinga, B. (1988). Modelling in KBS development. In *Proc. 2th European Knowledge Acquisition Workshop, Bonn, GMD-Studien 143*, pages 7.1–7.15, St. Augustin. GMD. Also in: Proc. 8th Expert Systems Workshop, Avignon, 1988.

Schreiber, G., Wielinga, B., & Akkermans, H. (1990b). Differentiating problem solving methods. Research Report ESPRIT Project P5248 KADS-II/T1.1/PP/UvA/002/1.0, University of Amsterdam & Netherlands Energy Research Foundation ECN. Submitted to IJCAI'91.

Schreiber, G., Wielinga, B., Hesketh, P., & Lewis, A. (1989b). A KADS design description language. ESPRIT Project 1098, deliverable B7 UvA-B7-PR-007, University of Amsterdam & STC Technology Ltd.

Schrijnen, L. & Wagenaar, G. (1988). Autopes: the development of an expert system for process control. In van Someren, M. & Schreiber, A., editors, *Proceedings First Dutch AI Conference NAIC'88*, pages 58–71, University of Amsterdam. Departmen of Social Science Informatics. In Dutch.

Shadbolt, N. & Wielinga, B. (1990). Knowledge based knowledge acquisition: the next generation of support tools. In Wielinga, B. J., Boose, J., Gaines, B., Schreiber, G., & van Someren, M., editors, *Current Trends in Knowledge Acquisition*, pages 313–338, Amsterdam. IOS Press.

Shortliffe, E., Scott, A., Bischoff, M., Cambell, A., van Melle, W., & Jacobs, C. (1981). Oncocin: An expert system for concology protocol management. In *AJCAI-81*, pages 876–881.

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

Tong, X., He, Z., & Yu, R. (1988). A survey of the expert system tool ZDEST-2. In *Proccedings ECAI'88, Munich*, pages 113–118, London. Pitman.

Ueberreiter, B. & Voß, A., editors (1991). *Materials KADS User Meeting, Munich, February 14/15 1991*. Siemens AG ZFE IS INF 32, Munich Perlach. In German.

van der Spek, R., van der Wouden, H., & Ysbrandy, C. (1990). The paint advisor. *Expert Systems*, 7(4):190–198.

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

Voß, A., Karbach, W., Drouven, U., & Lorek, D. (1990). Competence assesment in configuration tasks. In Aiello, L., editor, *Proceedings of the 9th European Conference on AI, ECAI'90*, pages 676–681, London. ECAI, Pitman.

Wielemaker, J. & Billault, J. (1988). A KADS analysis for configuration. ESPRIT Project P1098, Deliverable E5.1, University of Amsterdam.

Wielinga, B. & Bredeweg, B. (1988). Knowledge and expertise in expert systems. In van der Veer an G. Mulder, G., editor, *Human-Computer Interaction: Psychonomics Aspects*, pages 290–297, Berlin. Springer-Verlag.

Wielinga, B. & Breuker, J. (1984). Interpretation of verbal data for knowledge acquisition. In O'Shea, T., editor, *Advances in Artificial Intelligence*, pages 41–50, Amsterdam. ECAI, Elsevier Science publishers.

Wielinga, B. & Breuker, J. (1986). Models of expertise. In *Proceedings ECAI'86*, pages 306–318.

Wielinga, B., Schreiber, A., & Breuker, J. (1991). KADS: A modelling approach to knowledge engineering. ESPRIT Project P5248 KADS-II/T1.1/PP/UvA/008/1.0, University of Amsterdam. Submitted to *Knowledge Acquisition*.

Winkels, R. & Achthoven, W. (1989). Methodology and modularity in ITS design. In *Artificial Intelligence and Education*, pages 314–322, Amsterdam. IOS Press.

Wright, I., Hayball, C., Land., L., & Mulhall, T. (1988). Analysis report experiment f6. ESPRIT Project P1098, Deliverable E6.1, STC Technology Ltd. & Knowledge Based Systems Centre.

INTEGRATION OF KNOWLEDGE ACQUISITION TECHNIQUES WITHIN A GROUP DECISION SUPPORT SYSTEMS FRAMEWORK

M L G Shaw & B R Gaines

Knowledge Science Institute, University of Calgary
Calgary, Alberta, Canada T2N 1N4
(mildred@cpsc.ucalgary.ca, (403) 220-5114,
gaines@cpsc.ucalgary.ca, (403) 220-5901)

Abstract: Knowledge engineering for large scale systems is essentially a *group process* involving multiple requirements specifiers, knowledge engineers, domain experts, system implementation specialists, evaluation specialists and knowledgeable clients. Past research on knowledge acquisition tools has focused on the development of computer-based interactive systems to support *individual* technical activities in knowledge-based system development. Only recently has effort begun to be applied to support of the decision processes of the overall group participating in system development, particularly the interactions between group members. This paper reports work in progress on the extension of existing knowledge support tools, which already provide offline analysis of group-related issues such as conflicts in concepts and terminology, to provide online support of the group decision processes in both its social and technical aspects.

INTRODUCTION

Ever since Feigenbaum (1980) identified the knowledge engineering bottleneck impeding the effective application of knowledge-based systems, knowledge acquisition has been a topic attracting major research effort and resulting in the development of a wide range of methodologies, techniques and tools (Boose & Gaines, 1988, 1990; Gaines & Boose, 1988; Boose, 1989). In terms of the direct knowledge engineering process we now have a rich repertoire of approaches, support systems and experience on which to base further research and practical system development. However, the technological focus has been the support of individuals, such as tools to interview the expert or to support the knowledge engineer, and has neglected the essential social aspects of large-scale system development. As we become more and more concerned with issues of system requirements, integration with existing systems, validation and verification, documentation, enhancement and maintenance, it is becoming clear that the technological focus has to change from the individual to the group.

The next generation of knowledge support systems for large-scale knowledge-based system development has to be conceptualized within a group decision support systems (GDSS) framework. In particular, the separation between the decision support appropriate to system performance applications, and that appropriate to ongoing system development, appears increasingly inappropriate for knowledge-based systems. It is the nature of knowledge to develop through applications and to change continually in the light of reflection on experience. Development, validation and maintenance become a unitary process in knowledge-based systems.

The move towards group decision support has already begun in the knowledge acquisition community. Generic toolboxes encompassing and integrating a range of methodologies such as KSSO (Shaw & Gaines, 1987; Gaines, 1988) and AQUINAS (Boose & Bradshaw, 1987), what have come to be called second-generation knowledge acquisition tools (Gaines & Linster, 1990), already make provision for users with widely varying roles, requirements and capabilities. Liou, Weber and Nunamaker (1990) have

reported research on a methodology for knowledge acquisition in a group decision support system environment. Methodologies have also been developed that manage the elicitation of knowledge from groups of experts and are supported by tools that can analyze relations between the conceptual frameworks, and conflicts in the use of terminologies, for groups comprising multiple experts and clients (Gaines & Shaw, 1989; Shaw & Gaines, 1989).

We have previously emphasized the need to view knowledge acquisition as part of the socio-technical processes involved in knowledge flows in society (Shaw & Gaines, 1989), have developed socio-cognitive models of the way in which various forms of knowledge arise in individuals and society (Gaines, 1989a), and have extended our original concept of knowledge support systems (Shaw & Gaines, 1987) to encompass the wide variety of technologies supporting knowledge flows (Gaines, 1990c). This paper reports on our current research program for developing a group decision support infrastructure that integrates existing tools within a networked environment designed to support the group as a whole as well as individuals playing roles within it.

THE SOCIAL CONTEXT OF KNOWLEDGE ACQUISITION

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

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

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

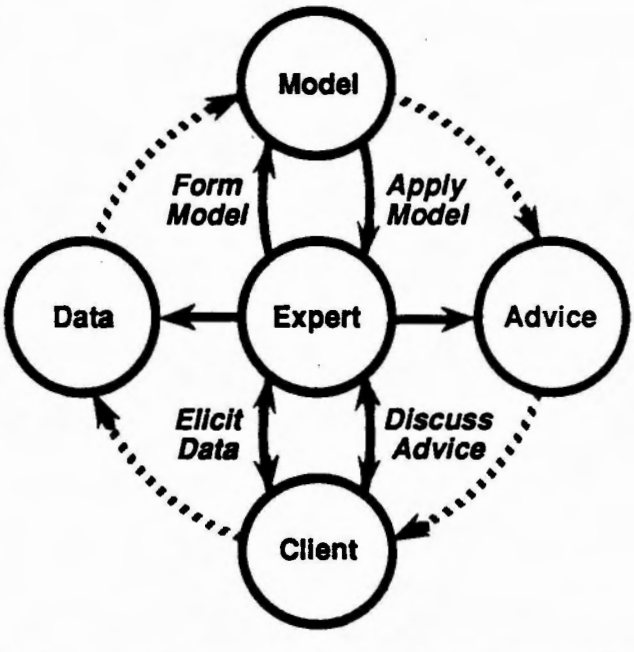


Figure 1 The negotiation cycle in expert-client interaction

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

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

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

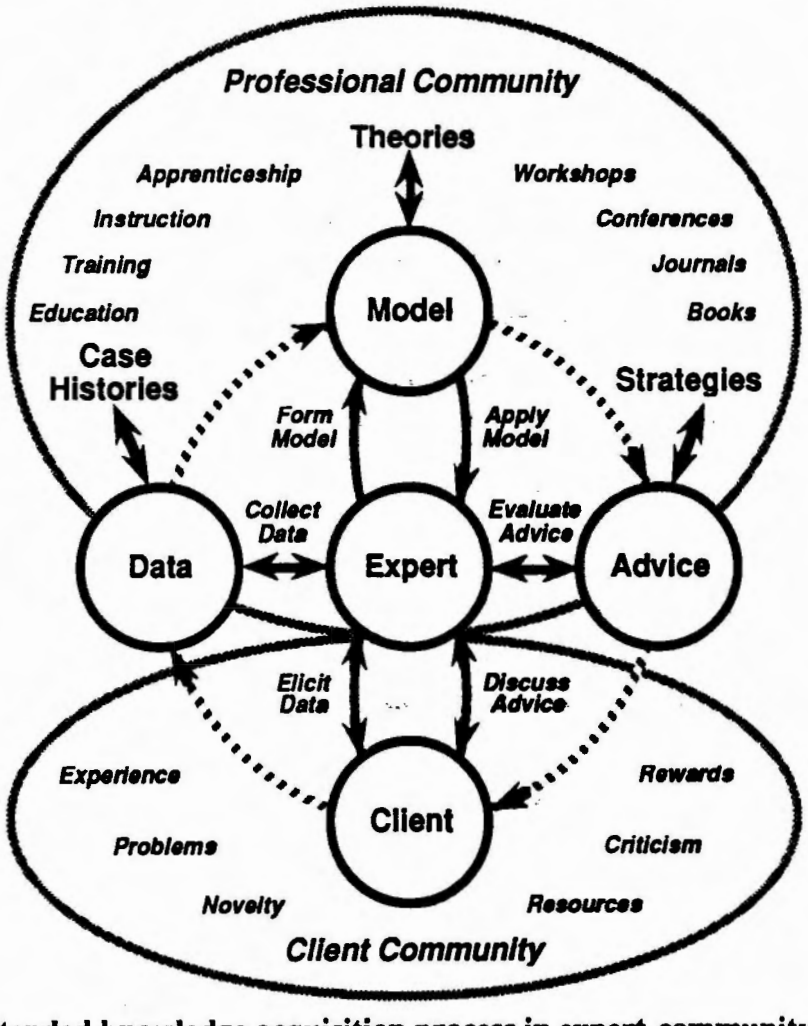


Figure 2 Extended knowledge acquisition process in expert-community interaction

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

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

In terms of design considerations for a next generation of knowledge acquisition tools, what the extended model of expertise in Figure 2 makes clear is that we should be making our objective the support of the knowledge processes of the *community* in which expertise plays a role. However, this does not necessarily mean that we have to discard the methodologies, techniques and tools that have been developed for use with individual experts. With the rapid pace of innovation in information technology, and the need for effective knowledge-based system development *now*, it is more realistic to design group support tools around the *integration* of existing techniques. In the longer term, the experience with such integration may lead to new designs that commence with a socially oriented paradigm, but we need much more practical experience before such designs can be proposed on a credible basis.

GROUP DECISION SUPPORT FOR KNOWLEDGE ACQUISITION

Figure 3 show an architecture for integrating existing knowledge acquisition techniques and tools to form a group decision support system for the knowledge processes of a community involved in knowledge-based system development. The roles shown in the lower part of the diagram are intended to be indicative rather than definitive. Users have to be supported in all these roles, and in hybrids and variants of them. Particular individuals may change between roles, either instantly as they operate the system or as part of changing capability and career migration, and hence it is important that the interface is as uniform as possible across roles.

The functionality shown in the boxes at the top of the diagram is also only illustrative, based on experience with existing individually oriented tools. It is reasonable to expect that needs for new functionality will become apparent as we gain experience of the group decision support approach. The center box already contains functionality such as email and conferencing that is not normally integrated with a knowledge acquisition system and yet is clearly part of the support of group processes. To what extent is it possible to use existing email and conferencing, and how much modification is required to achieve effective integration?

It is important to note that integration is itself a structured concept. System integration at the level of inter-application communication is easy to achieve. System integration at the level of appropriate mutual control of functionality and data exchange is also largely an implementation issue. However, knowledge level integration such as that necessary to support the combination and editing of knowledge structures in several different tools with different perspectives is very much more difficult.

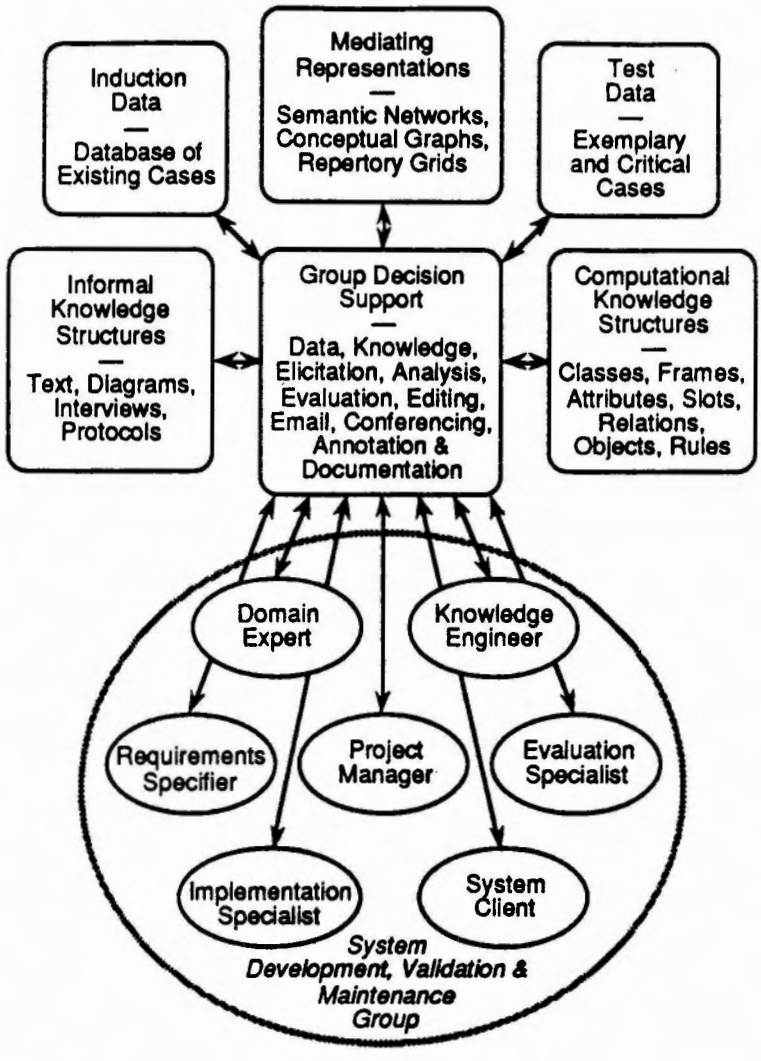


Figure 3 Group decision support system for knowledge-based systems development

Thus, Figure 3 presents an intermediate stage in system development, not an ultimate architecture. Over the next few years experience in the design, implementation and application of systems such as that shown will begin to generate the examples and principles necessary to the well-founded design of group decision support systems for the development of large scale knowledge based systems. Meanwhile, the considerations discussed above provide guidelines for research and development.

EXISTING TOOLS WITHIN THE GDSS CONTEXT

The architecture we have adopted for GDSS developments is the heterogeneous integration of existing tools, involving substantial redevelopment but not the design and implementation of a new monolithic system. This is an important issue in its own right and our logic for heterogeneous integration is manyfold (Gaines, 1990a); notably: the rapid pace of change of all the base technologies making systems obsolescent as they are implemented; the need to incorporate subsystems optimized for particular roles, probably by different organizations; and the overarching requirement for continuous upgrading and enhancement without a total system rebuild. All our current system development is based on highly modular systems communicating through servers and implemented as class libraries in object-oriented languages (Gaines, 1991b).

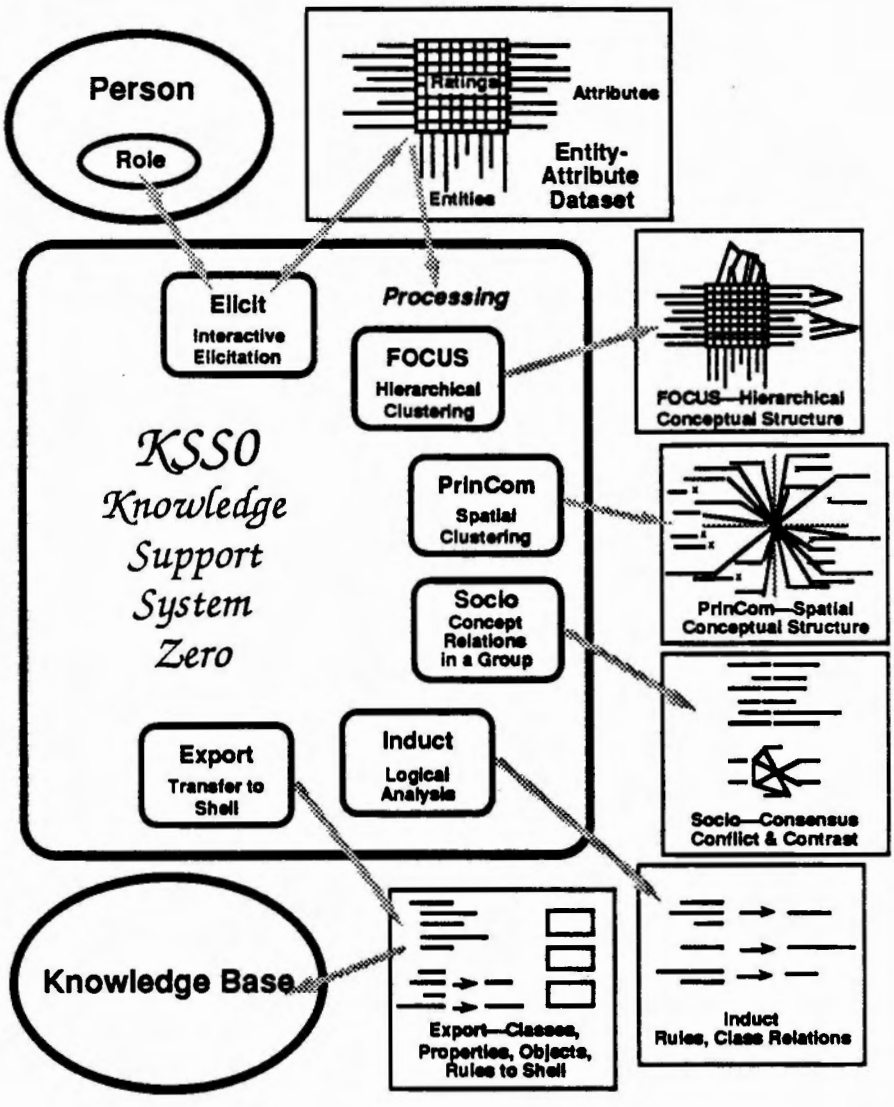


Figure 4 Architecture and facilities of KSS0, a knowledge acquisition workbench

Our original knowledge acquisition tool, KSS0 (Shaw & Gaines, 1987; Gaines, 1988) shown in Figure 4, providing repertory grid elicitation, conceptual clustering, conceptual comparison, empirical induction, and knowledge base creation tools, is already configured as a set of modules around a specialist database. KSS0 focuses on mediating representations supporting the expert's modeling processes in moving from skilled performance to overt knowledge structures supporting its emulation in the computer. In recent years we have extended KSS0 to support the informal representations of knowledge that are prior to those within the tool, such as text, pictures, diagrams, semi-structured interviews, protocols, and so on (Gaines, Rappaport & Shaw, 1989; Gaines & Linster, 1990). This has been done by providing an interapplication protocol allowing KSS0 to interact with Apple's HyperCard to provide the appearance of a seamless single application to users. KSS0-specific functionality in HyperCard is supported by scripts that allow conceptual structures on the KSS0 side to be linked to informal sources and annotation on the HyperCard side. However, the system is designed to provide an open architecture so that domain-specific tools may be developed in HyperCard and existing knowledge acquisition tools in HyperCard may be integrated, for example Woodward's (1990) Cognosys for the analysis of protocol data in textual form.

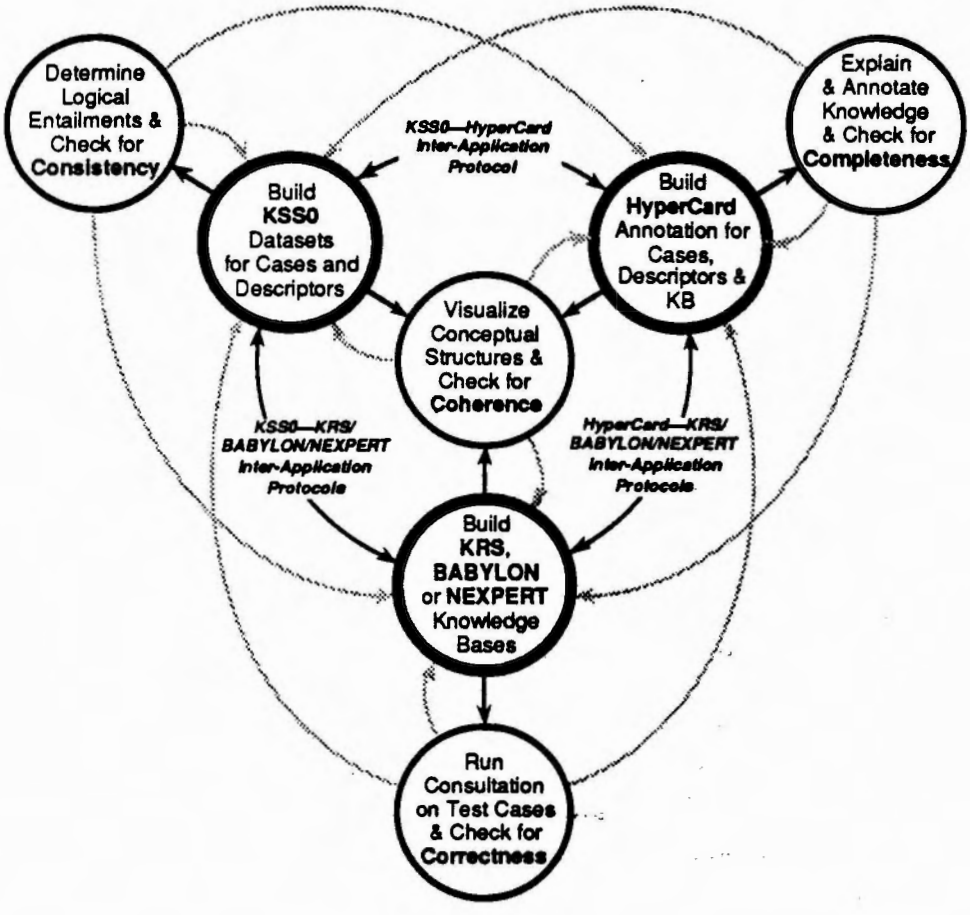


Figure 5 Heterogeneous integration of knowledge acquisition, hypermedia and knowledge-based system shells

KSS0 exports knowledge bases to a range of existing knowledge-based system shells, and we have reported a number of collaborative studies in which we have integrated these shells directly with KSS0 and HyperCard through inter-application protocols (Gaines, Rappaport & Shaw, 1989; Gaines & Linster, 1990). This leads to new modes of knowledge acquisition in which, for example, a test case that leads to an incorrect decision within the performance engine is posted directly back to KSS0 where the induction tool is automatically rerun to update the knowledge base taking into account the new test case.

The heterogeneous integration of knowledge acquisition, hypermedia and performance tools leads to a variety of options for validation and verification at each stage of the system development. Figure 5 illustrates the system architecture and classifies the forms of validation supported.

At the center of Figure 5, it is shown that each of the tools provides means for *visualizing* the conceptual structures they support. For example, KSS0 provides various clustering algorithms for presenting repertory grid data graphically, HyperCard is essentially visual, and KRS (Gaines, 1990b, 1991a), BABYLON (Christaller, di Primio & Voß, 1989) and NEXPERT (Rappaport, 1987) each provide their own knowledge-base graphers. From a formal validation point of view such visualization supports the expert and knowledge engineer in evaluating the *coherence* truth of the knowledge structures—their internal relations and consistency. Such evaluation provides feedback through each of the tools to improve expressiveness and correct errors that are made manifest through incoherence.

At the top left of Figure 5, the induction tool in KSS0 derives plausible constraints on the database of case data (essentially empirical rather than normative normalization of the database) in the form of entailments that eventually become expressed as rules. It supports the evaluation of these entailments for mutual *consistency* (can two rules arrive at different conclusions on the same case) and for consistency with the case data. Such evaluation again provides feedback through all the tools—for example, one may amend cases in KSS0, amend rules in KRS, or make a note of the potential problem in HyperCard.

At the bottom of Figure 5, the running of further cases, perhaps in routine system use, provides evaluation of the *correctness* of the knowledge base as a decision support tool. Again such evaluation provides feedback through all the tools, amendment of the KSS0 case base and re-induction, direct editing of the knowledge structures in KRS, or commentary in HyperCard.

Finally, at the top right of Figure 5, it is shown that the hypermedia tool provides a far more significant logical validation role that its annotation duties may suggest. Systems intrinsically cannot validate themselves for *completeness*, and clearly there can be no guarantees of completeness in an open universe. However, in terms of validating the formal knowledge base completeness, the informal knowledge base held in hypermedia form plays a very significant role. Anything mentioned informally that has no referent formally leads to a suspicion of incompleteness requiring further investigation.

INTEGRATING THE TOOLS WITHIN A GDSS INFRASTRUCTURE

Most of the technical functionality required for group decision support at the knowledge base level is already present in the existing systems of Figure 5. The tools each support various modes of user interaction for different user roles, and provide communication and data transformation protocols supporting mutual integration. What is missing, however, is explicit recognition within the existing technologies of the group as an entity, and technical support for the management of group processes.

In a different context we have developed a special-purpose 'groupware' system based on the integration of the repertory grid components of KSS0 (Shaw, 1980; Shaw & Gaines, 1983) with an electronic mail sub-system to facilitate the formation and management of 'special interest groups' (Shaw & Gaines, 1991a). The architecture of this system, *RepGrid-Net*, is shown in Figure 6. RepGrid-Net uses the Socio analysis tools within KSS0 to analyze and display graphically the comparisons between individual conceptual structures, and socionets of a complete group showing the extent to which one individual has the concepts to understand another.

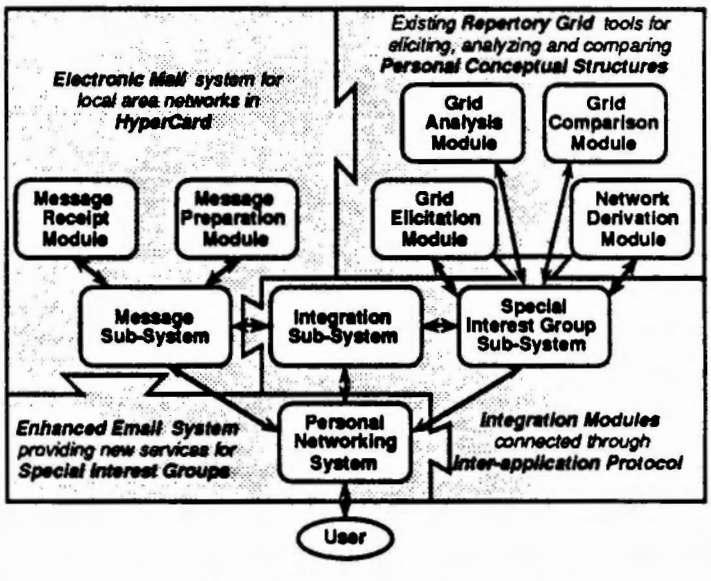


Figure 6 RepGrid-Net integration of repertory grid elicitation and analysis with electronic mail to facilitate the formation and management of special-interest groups

The development and application of RepGrid-Net showed the major system impact that the addition of facilities for electronic mail may have on an interactive system. Activities such as annotation and documentation are notoriously difficult to encourage in a development team. Part of the problem is that the underlying human process is one of discourse—what is being produced is best regarded as messages to other people not as an abstract text database. Our preliminary studies suggest that users may more readily add such information if there is a specific audience as there is in the human social situation.

A second parallel development of a technology that, at first sight, appears to have little to do with knowledge acquisition has been our development of *GroupWriter*, a word processor to support collaborative authoring of scientific papers (Malcolm & Gaines, 1991). GroupWriter appears to its users very much like Microsoft Word but gives access to alternative versions of a document, and text and speech annotation, through a narrow side bar in which paragraphs with alternates or annotation are bracketed. Moving the cursor into this side bar gives access to popup menus allowing alternates to be shown and merged, and annotation to be shown or played.

Since GroupWriter provides a hypertext-style linkage structure to document components it is readily extended to provide hypertext links to modules within a knowledge acquisition system. For example, source documents and transcripts of interviews and protocols may be linked to the knowledge structures that have been developed based on them. This can be done without undermining the visual appearance of the original document. Documents generated as part of the knowledge engineering process can also be treated in this way—they are normal documents with full typographic formatting, embedded pictures, and so on, but they are also tightly embedded in the knowledge structures being developed.

Most of RepGrid-Net is written in HyperCard which has been the basis of our integration of KSS0 with NEXPERT and BABYLON as shown in Figure 5, and GroupWriter is written as modules in the same class library in C++ as our own knowledge-based system shell, KRS. Hence, it has been technically simple to link them both into our existing knowledge acquisition tools. Figure 7 shows the current state of development of KSSn, the general knowledge support system that is being developed as a successor to KSS0.

At the center is the group decision support system seen by the user community, provided by integrating the sub-systems in the outer ring through an inter-application communication protocol. Going around the diagram clockwise from the bottom left:

- The current *electronic mail* system is the HyperCard/AppleTalk system developed for RepGrid-Net. This has been useful for laboratory trials and demonstrations but, for significant applications, is being extended to interface to the full Internet mail system.
- The *group writing* system has been discussed above. This is also being extended currently to operate across Internet, and facilities are being added to import and export documents from other text processing systems supporting RTF or SGML.
- The current *hypermedia* system is the HyperCard/SuperCard system developed for informal knowledge base development and knowledge base annotation as part of the integration with NEXPERT and BABYLON. The script languages of the hypertext systems give this module far greater significance than is apparent in the figure. It is the most open architecture and customizable component of the system and, hence, is the natural vehicle for further integration with other systems.
- The *text analysis* tools are currently the Texan lexical and clustering modules from KSS0. They have been rewritten in C++ to export text clusters as knowledge structures in the visual language of KRS. The current text analysis is useful as a vocabulary acquisition tool for the other elicitation techniques. In the longer term the text analysis system needs to be extended with the more powerful parsing and semantic analysis techniques now being developed for knowledge bases (Gomez & Segami, 1990).

- The *knowledge elicitation* tools consist of the modules of KSS0 shown in Figure 4. The strength of this system is the ease of use of its graphic elicitation and clustering tools by end users such as domain experts. Its weakness is the limited representational power of the entity-attribute database structure of basic repertory grids. These were overcome in AQUINAS through extensions to hierarchical structures, and we have adopted a KL-ONE-like knowledge representation schema in KRS to provide similar extensions for KSSn. However, developing user interfaces which maintain the usability of KSS0 is problematic with the new data structures and, whilst these are being developed and evaluated, the original KSS0 system remains an important resource.
- The *empirical induction* tools are based on Induct (Gaines, 1989b), a system extending Cendrowska's (1987) Prism algorithm to develop rules directly from noisy datasets.
- The *visual language* tools are based on an interactive graphical structure edit for concept maps and semantic nets (Gaines, 1991c). They support the automatic layout of existing KL-ONE-like knowledge structures, and the interactive entry and editing of such knowledge structures. They compile them to the intermediate languages of a number of knowledge-based system shells.
- The *knowledge-based system* shells currently supported are KRS, NEXPERT and BABYLON. These are sufficiently diverse to force us to maintain an open architecture at this interface which should be extensible to other knowledge-based system shells.

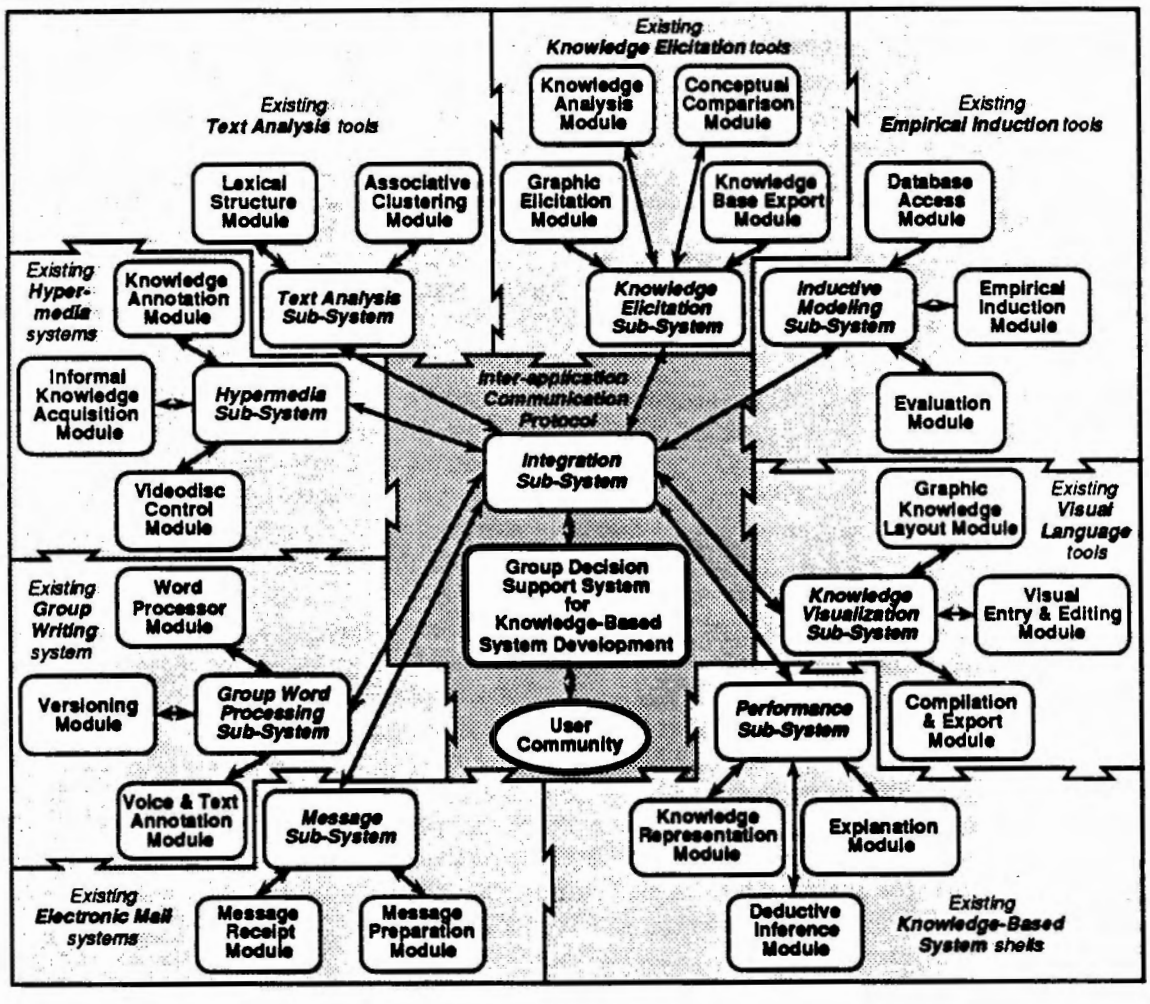


Figure 7 Group decision support system for knowledge-based systems development

All of the components of the system of Figure 7 are now in operation in a variety of different configurations. It is not clear whether the system as shown will ever, or need ever, exist as an ostensible monolithic piece of software or a particular workstation and network configuration. This is an important feature of the architecture, that it is open and reconfigurable for particular studies, developments and applications. In particular, it is designed such that tools from other research groups and commercial vendors may be incorporated with the minimum of effort.

The complexity and diversity of technologies involved in Figure 7 makes clear the need, in research at least, for the heterogeneous integration approach discussed earlier. The various sub-systems have to be developed as modular, reliable and usable systems in their own right. For example, the electronic mail and group writing sub-systems need to be proven as viable mailing and writing systems in routine use before they are incorporated in the overall system. Little can be learnt from studies of integrated systems if critical sub-systems are not effective when used alone. The complexity can also become a major impediment to development if the effort of modifying the system becomes too great. It is important to be able to develop different types of system with such little effort that there is no resistance to discarding them when they have served their experimental function.

It is also worth noting the sub-systems that are *not* included in Figure 7:

- *Conferencing* has been omitted because the problem of managing real-time interaction between participants in a conferencing situation is still a major research problem. We believe that much can be achieved through non-real-time interaction through email, documents and knowledge structures, but conferencing is an obvious technology to eventually include in experiments with knowledge acquisition in a group decision support framework.
- *Tutoring/training* has been omitted as a specific module because we see the knowledge support system of Figure 7 as providing an adequate framework for investigating applications to education without any specific tutoring tools. Some preliminary experiments on the HyperCard/KSS0/KRS combination in supporting graduate student's creativity have been very encouraging and suggest that knowledge acquisition toolkits have important educational roles (Shaw & Gaines, 1991b).
- *Analogical* and *case-based* reasoning modules have been omitted because we have not had time to develop significant sub-systems yet. However, the direct comparison of knowledge structures, abstraction from them, and inference using case data are clearly important extensions to the reasoning mechanisms offered by the shells already included.

Others will find additional components missing. This seems inevitable. It seems unlikely that any knowledge support system will ever be complete. One can only attempt to provide an extensible architecture of sufficient generality to allow enhancement, and even the architecture must eventually become obsolete as our knowledge and requirements develop.

CONCLUSIONS

This paper has reported ongoing research on group decision system support architectures for knowledge based system development. It has been argued that knowledge engineering for any realistic system is essentially a group process involving multiple requirements specifiers, knowledge engineers, domain experts, system implementation specialists, evaluation specialists and knowledgeable clients. This paper has reported the extension of our existing knowledge support tools to provide online support of the group decision processes in both its social and technical aspects. In particular we have integrated electronic mail and conferencing facilities with the existing knowledge acquisition tools, hypermedia systems and knowledge-based systems shells.

ACKNOWLEDGEMENTS

This work was funded in part by the Natural Sciences and Engineering Research Council of Canada. We are grateful to many colleagues who have played significant roles in exchanging ideas and research data with us that have been very important to the developments reported here, notably, John Boose, Jeff Bradshaw, Bill Clancey, Paul Compton, Bob Jansen, Marc Linster, Ross Quinlan, Alain Rappaport, and Brian Woodward, plus many stimulating and influential discussions at the Knowledge Acquisition Workshops worldwide.

REFERENCES

Boose, J H. (1989). A survey of knowledge acquisition techniques and tools. **Knowledge Acquisition 1** (1), 39-58 (March).

Boose, J.H. & Bradshaw, J.M. (1987). Expertise transfer and complex problems: Using AQUINAS as a knowledge acquisition workbench for knowledge-based systems. **International Journal of ManMachine Studies, 26,** 328.

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

Boose, J.H. & Gaines, B.R., Eds. (1990). **The Foundations of Knowledge Acquisition.** London, Academic Press.

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

Christaller, T., di Primio, F. and Voß, A. (1989) **Die KI-Werkbank BABYLON.** Bonn: Addison Wesley.

Feigenbaum, E.A. (1980). Knowledge Engineering: the Applied Side of Artificial Intelligence. **Report STAN-CS-80-812.** Department of Computer Science, Stanford University.

Gaines, B.R. (1988). Knowledge acquisition systems for rapid prototyping of expert systems. **INFOR, 26**(4), 256-285 (November).

Gaines, B.R. (1989a) Social and cognitive processes in knowledge acquisition. **Knowledge Acquisition 1**(1), 251-280 (March).

Gaines, B.R. (1989b) An Ounce of Knowledge is Worth a Ton of Data: Quantitative Studies of the Trade-Off between Expertise and Data based on Statistically Well-Founded Empirical Induction. **Proceedings of 6th International Workshop on Machine Learning,** pp.156-159. San Mateo, California: Morgan Kaufmann (June).

Gaines, B.R. (1990a). Organizational integration: modeling technology and management. Zunde, P. & Hocking, D. (Eds.) **Empirical Foundations of Information and Software Sciences II.** pp. 51-64. New York: Plenum Press.

Gaines, B.R. (1990b) Knowledge representation servers: a generic technology for knowledge acquisition and knowledge-based systems. **Proceedings of the First Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop: JKAW'90.** pp. 413-430. Tokyo: Ohmsha (October).

Gaines, B.R. (1990c). Knowledge support systems. **Knowledge-Based Systems** 3(4) 192-203 (December).

Gaines, B.R. (1991a). Empirical investigation of knowledge representation servers: design issues and applications experience with KRS. **AAAI Spring Symposium: Implemented Knowledge Representation and Reasoning Systems.** pp. 87-101. Stanford (March).

Gaines, B.R. (1991b). Object-oriented implementation of object-oriented knowledge representation. **Proceedings of the Third AAAI Workshop on Object-Oriented Programming in AI.** Anaheim, Califonia (July).

Gaines, B.R. (1991c). An interactive visual language for term subsumption visual languages. **IJCAI'91: Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence.** San Mateo, California: Morgan Kaufmann.

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

Gaines, B.R. & Linster, M. (1990). Development of second generation knowledge acquisition systems. Wielinga, B., Boose, J.H. & Gaines, B.R., Schreiber, G., van Someren, M., Eds. **Current Trends in Knowledge Acquisition.** pp. 143-160. Amsterdam, IOS.

Gaines, B.R., Rappaport, A. & Shaw, M.L.G. (1989). A heterogeneous knowledge support system. Boose, J.H. & Gaines, B.R., Eds. **Proceedings of the Fourth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp.13-1-13-20. Banff (October).

Gaines, B.R. & Shaw, M.L.G. (1989). Comparing the conceptual systems of experts. **Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.** pp. 633-638. San Mateo, California: Morgan Kaufmann (August).

Gomez, F. & Segami, C. (1990). Knowledge acquisition from natural language for expert systems based on classification problem-solving methods. **Knowledge Acquisition,** 2(2), 107-128 (June).

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

Liou, Y.I., Weber, E.S. & Nunamaker, J.F. (1990). A methodology for knowledge acquisition in a group decision support environment. **Knowledge Acquisition 2** (2), 129-144 (June).

Malcolm, N. & Gaines, B.R. (1991). A word processor supporting group-writing. **Technical Report No.91/420/04.** Department of Computer Science, University of Calgary (February).

Rappaport, A. (1987) Multiple-problem subspaces in the knowledge design process. **International Journal of Man-Machine Studies 26**(4), 435-452 (April).

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

Shaw, M.L.G. (1980). **On Becoming A Personal Scientist: Interactive Computer Elicitation of Personal Models Of The World.** London: Academic Press.

Shaw, M.L.G. & Gaines, B.R. (1983). A computer aid to knowledge engineering. **Proceedings of British Computer Society Conference on Expert Systems,** 263-271 (December). Cambridge.

Shaw, M.L.G. & Gaines, B.R. (1987). KITTEN: Knowledge Initiation & Transfer Tools for Experts & Novices. **International Journal of Man-Machine Studies, 27,** 251-280.

Shaw, M.L.G. & Gaines, B.R. (1989). A methodology for recognizing conflict, correspondence, consensus and contrast in a knowledge acquisition system. **Knowledge Acquisition 1**(4), 341-363 (December).

Shaw, M.L.G. & Gaines, B.R. (1991a). Supporting personal networking through computer networking **Proceedings of CHI'91: Human Factors in Computing Systems.** New York: ACM Publications.

Shaw, M.L.G. & Gaines, B.R. (1991b). Mapping creativity with knowledge support tools. **Proceedings of the AAAI Workshop on Creativity: Models, Methods and Tools.** Anaheim, California (July).

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

Woodward, B. (1990) Knowledge engineering at the front-end: defining the domain. **Knowledge Acquisition, 2**(1) 73-94 (March).

Use of the KADS knowledge-acquisition method to model the ONCOCIN task of cancer-chemotherapy administration

Marc Linster
Expert System Research Group
GMD
PO Box 1240
D-5205 St. Augustin 1
FRG

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

ABSTRACT

Builders of expert systems have become increasingly interested in the KADS method of knowledge acquisition in recent years. Despite the apparent face validity of the KADS approach, reports of the use of KADS to develop large application systems are virtually nonexistent. We used KADS to model the task of cancer-chemotherapy administration performed by ONCOCIN, a large medical expert system developed at Stanford University in the 1980s. Based on a knowledge level description of ONCOCIN and on a review of the features of KADS, we developed K-ONCOCIN, a model of the ONCOCIN cancer-chemotherapy task that can be understood in terms of the four layers required by the KADS approach.

In this paper, we offer a detailed description of the elements and layers of the K-ONCOCIN conceptual model. We also describe briefly a design model that implements K-ONCOCIN as a functional expert system. The K-ONCOCIN model provides a data point for understanding the strengths and limitations of the KADS method when applied to a complex domain task.

1. INTRODUCTION

Workers in artificial intelligence (AI) have used a variety of models to analyze successful expert system applications (Clancey, 1985) and to provide the foundations for future knowledge engineering (McDermott, 1988). Combined with algorithms and data-structures, these models can provide structuring mechanisms for the representation and use of knowledge (Chandrasekaran, 1988). Models and their application to knowledge engineering have become topics of active research (Musen 1989a; Karbach, Linster & Voß, 1990). The diversity of the modeling approaches that are discussed in the literature (for example, Clancey, 1985; Alexander et al., 1987; Breuker et al., 1987; Shaw & Gaines, 1987; Chandrasekaran, 1988; Marcus, 1988; McDermott, 1988; Musen, 1989c; Boose, Shema & Bradshaw, 1989; Gruber, 1989; Punch & Chandrasekaran, 1990; Gaines & Shaw, 1990; Bredeweg et al., 1989) calls for a systematic comparison using common exemplars that can demonstrate how the different approaches relate to one another. In this paper, we use the Knowledge Acquisition and Design Structuring (KADS) methodology developed at the University of Amsterdam (Wielinga & Breuker, 1986; Breuker et al., 1987; Schreiber et al., 1988; Breuker & Wielinga, 1989) to model the task of cancer-chemotherapy administration—the same application task modeled by the ONCOCIN expert system (Shortliffe et al., 1981; Tu et al., 1989).

The KADS approach is a promising set of modeling techniques for developing commercial expert-system applications (for example, Schachter-Radig & Wermser, 1988) that also provides the scientific foundation for much current research (Voß et al., 1990). Previous publications on KADS either have focussed on very limited application problems to illustrate the methodology (Wielinga et al., 1989) or have discussed the approach on a rather abstract level (Breuker et al., 1987). To provide a sound data point for an evaluation of KADS, we selected ONCOCIN as a reference task. ONCOCIN is a real-life application that has been the basis for considerable work in the area of knowledge acquisition and that has been broadly described and discussed in the literature.

We shall first give a knowledge-level overview of ONCOCIN. We then provide a brief summary of those aspects of the KADS methodology that are relevant for this paper. Using the terminology of KADS, we then describe K-ONCOCIN—a KADS model of the ONCOCIN task—which forms the basis for our evaluation and discussion of the KADS approach.

2. A KNOWLEDGE-LEVEL ANALYSIS OF ONCOCIN

We first provide an overview of the knowledge-level (Newell, 1982) components and behaviors of ONCOCIN. We analyze ONCOCIN from the perspective of OPAL (Musen, 1987), a knowledge-acquisition tool that facilitates development of the ONCOCIN knowledge base. Our emphasis is on the goals, actions, and knowledge structures of the ONCOCIN system. A technical description of the ONCOCIN implementation has been published elsewhere (Tu et al., 1989).

2.1. Overview

ONCOCIN provides an automated mechanism to monitor the treatment of cancer patients assigned to standard treatment protocols. It includes a database of specifications for a number of protocols and a database of test and treatment records for a number of patients. The system assists the treating physician by providing recommendations for patient therapy and for the requisition of tests according to protocol guidelines. On the basis of (1) the latest laboratory-test results for a particular patient, (2) information from the most recent clinical examination, (3) the patient's treatment history, and (4) the specifics of the treatment protocol to which the patient is assigned, ONCOCIN computes a recommendation for the next cycle of treatments and tests for that patient.

We shall now give a detailed account of the protocol knowledge that ONCOCIN uses to generate its recommendations. This knowledge bears directly on the design decisions for the KADS model of ONCOCIN. Because K-ONCOCIN models a complex application task, we must present a considerable amount of domain-specific knowledge to define the salient features of our model.

2.2. The specification of a protocol

A treatment protocol is defined by two kinds of information:

(1) Each protocol has a time- and decision-oriented control structure that determines all permissible sequences of tests and treatment cycles (Section 2.2.1).

(2) Each test and treatment sequence referred to by the control structure is specified in detail (Section 2.2.2).

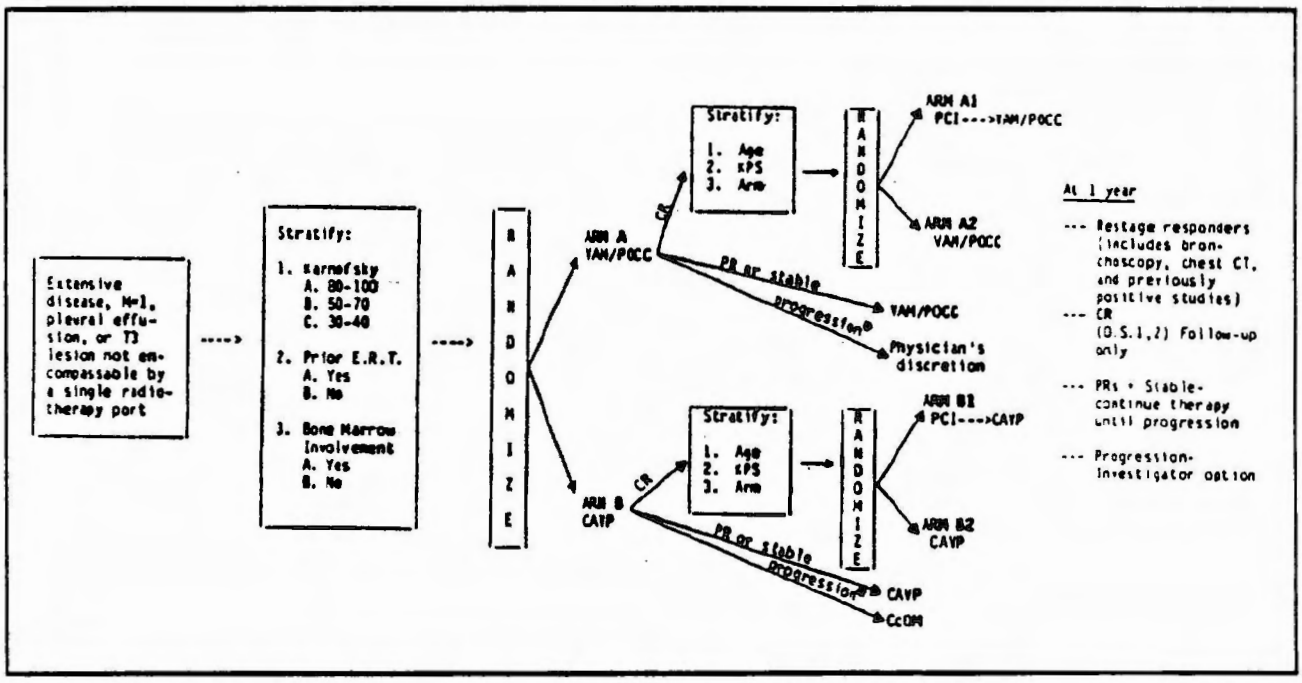


Figure 1: The schema of Northern California Oncology Group protocol 2083-1 to treat small-cell lung carcinoma. Patients must meet certain preconditions to be enrolled in the protocol. They are stratified according to their Karnofsky status, prior emergency radiation therapy and bone-marrow involvement. They are then randomly assigned to arms A or B of the protocol. Arm A analyzes the combination of the chemotherapies VAM and POCC and the effect of prophylactic cranial irradiation (PCI). Arm B tests the chemotherapies CAVP and CcOM and the effect of PCI.

2.2.1. The Control Structure of a Protocol

For an individual patient, a protocol treatment consists of a sequence of tests, possible X-ray therapies (XRT), and possible chemotherapies (chemo). The control structure of the protocol determines all permissible sequences of tests and treatments for any patient enrolled in the protocol (Figure 1). A variety of sequences are provided by the incorporation into the control structure of branching logic that depends on the results of particular tests or clinical assessments, and by branch points that represent situations in which patients are assigned at random to alternative forms of treatment as part of the clinical experiment embodied in the protocol.

The diagram in Figure 1 describes the general schema of Northern California Oncology Group protocol 2083-1 in terms of chemotherapies, X-ray therapies, tests, and randomizations (Carlson et al., 1986).

2.2.2. Specification of Tests and Treatments

Each protocol includes a list of the initial laboratory tests and lists of the tests that are to be repeated during and after the treatment sequence. The information about each repeated laboratory test includes the frequency of testing and various disease- and treatment-related conditions that indicate (or contraindicate) the appropriateness of the test.

An X-ray therapy is specified by the total radiation dosage, the number of patient visits, the dosage at each visit, and the period of time over which the therapy is to be administered. If the X-ray therapy is to interrupt a chemotherapy, the point of interruption is specified.

A chemotherapy is specified as (1) a number of cycles and sub-cycles of various drug treatments and (2) the laboratory tests to be administered on specific days of each cycle. (A cycle is simply a particular course of drug treatment administered for a particular number of days.) In cases of severe drug toxicity, of abnormal laboratory-test results, or of abnormal clinical symptoms, the specification includes details of treatments to be administered during delays to (or on termination of) the standard treatment cycles. Also specified are both the time that must be allowed between the start of the therapy and the end of any prior X-ray therapy, and the maximum number of consecutive treatment delays that may occur during the therapy.

The specification of a drug treatment indicates, for each drug in the treatment, the normal dose (either expressed in absolute terms or in relation to some patient characteristic such as body weight), the route of administration, the duration of the administration of one dose, the number of administrations, the frequency of administrations, the days in the cycle on which administrations are to be made, the maximum individual dose, and the maximum cumulative dose. In addition, the specification indicates actions to be taken if the degree of drug toxicity is too high, if the patient's white-blood-cell count or platelet count fall within certain ranges, or if other specific laboratory-test results are abnormal. The actions applicable in such circumstances may include attenuation of the dosage of some drug(s), substitution of another specified drug treatment, a specified break in treatment, termination of the current cycle (in which case a special drug treatment may be used before the start of the next cycle), a skip to the next cycle (for alternating chemotherapies), a switch to another named protocol, removal of the patient from any treatment protocol, reporting the result to the person who developed the protocol, and consultation with other physicians. Protocols also may specify dosage-attenuation rules for the use of drugs in cycles that *follow* delayed and aborted cycles, and for the administration of drugs subsequent to X-ray therapy.

A rule for attenuating the dosage of a drug may express the recommended dose either as a percentage of the normal dose or as a percentage of the previous (possibly already attenuated) dose. Whenever more than one attenuation rule may be applicable, there are rules to specify whether the recommended attenuation should be based on the combination of factors (by multiplying the attenuation fractions) or whether a single attenuation should be applied. For each drug in each treatment, there may be specified a minimum attenuation percentage that represents the level at which it is more appropriate to suspend the cycle than to continue reducing the dosage. The indication for such a delay will be overridden if the maximum number of delays for the therapy has already been reached. In that case a minimum default attenuation is applied. A special rule is used to round off the calculated dose to an appropriate unit for each specific drug (e.g., tablet or vial size).

2.3. The Patient Database

Information stored for each patient includes the name of the assigned protocol and a time-stamped record of each clinic visit. A visit record contains both the results of laboratory tests and of clinical examinations conducted since the previous visit and an account of X-ray or drug treatments administered at the recorded visit.

3. KADS

The KADS (Knowledge Acquisition and Design Structuring) methodology has been developed in the ESPRIT Project P1098 to provide methodological support for the industrial development of knowledge-based systems. We

give a short review of the methodology[1]. As KADS is still under development, the following section is certainly influenced by our backgrounds and by our personal interpretations of aspects of the KADS approach.

3.1. The Methodology

KADS is a methodology for the structured and systematic development of knowledge-based systems, which aims to provide software engineering support for the knowledge engineering process—from the first exploratory interviews of experts to the development of an operational system. KADS is a manual, non-formal method. Work is underway to provide a KADS workbench of computer-based tools that will assist knowledge engineers in applying the KADS principles (Anjewierden & Wielemaker, 1990).

KADS divides the development process of a knowledge-based system into a suite of models that represent the transformation of knowledge from the initial, non-formal phases into an operational system. Within the models, knowledge is represented on several layers to stress the different types of expertise that developers must be represent within a knowledge based system.

3.2. The Layers of Knowledge in KADS

In all KADS models, four types of knowledge may be distinguished: domain-layer, inference-layer, task-layer and strategy-layer knowledge. Figure 2 shows the four layers and how they interact.

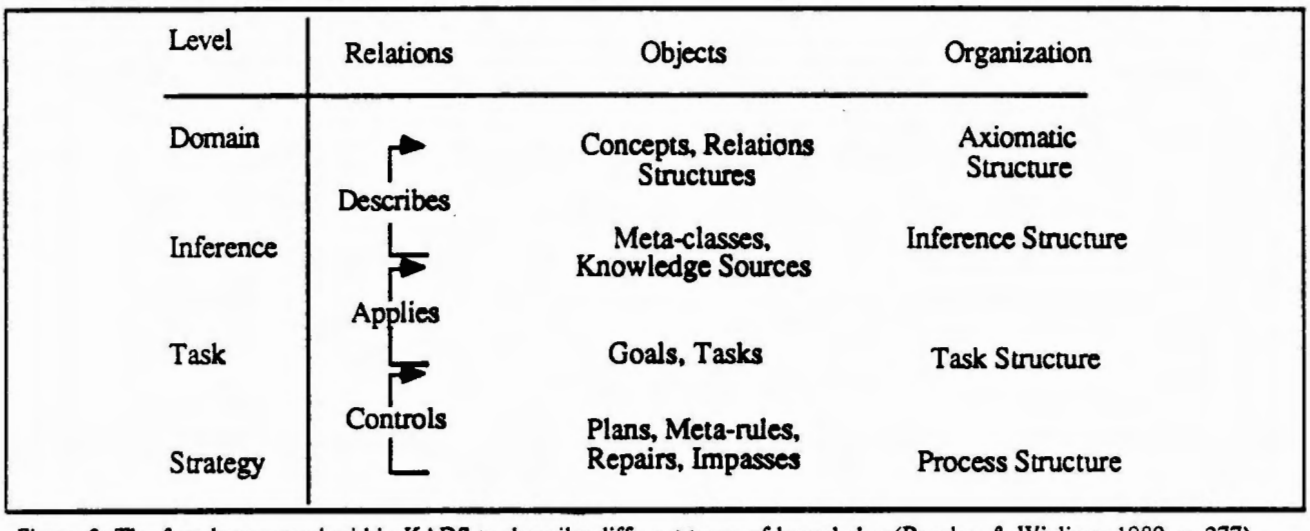


Figure 2: The four layers used within KADS to describe different types of knowledge (Breuker & Wielinga 1989, p. 277).

Within each layer, KADS provides the vocabulary for the description of the knowledge elements of the layer. KADS provides few pre-defined description elements for the layers. We will illustrate the layers of KADS with examples of ONCOCIN applications.

3.2.1. The Domain Layer

The *domain layer* describes all the knowledge of the application domain in the form of concepts, structures, and relations. The representation of the domain-layer knowledge is independent of the inferences that are to be made with the domain-layer knowledge by the problem solver.

In the domain of cancer treatment, for example, the drug *VP-16*—along with its dosage, route of administration, and maximum cumulative dose—is defined as a domain-layer concept. The knowledge that *VP-16* is a part of the chemotherapy *VAM* can be represented as a relation.

3.2.2. The Inference Layer

The *inference layer* describes the problem-solving capabilities of a system in terms of *knowledge sources* and *meta-classes*. Meta-classes describe roles and states in the problem-solving process. Knowledge sources are functional de-

---

[1]We will not consider the very recent developments of KADS in the REFLECT project (Akkermans et al. 1990, Voß et al 90). These developments are forming the basis for reflective competence assessment of knowledge based systems.

scriptions of inferences that operate on meta-classes. The combination of meta-classes and knowledge sources forms a domain-independent data-dependency diagram: the *inference structure*. Figure 3 shows a taxonomy of knowledge sources proposed by the developers of KADS as a basis for representing an initial set of interpretation models (Breuker et al., 1987, pp. 36 - 40).

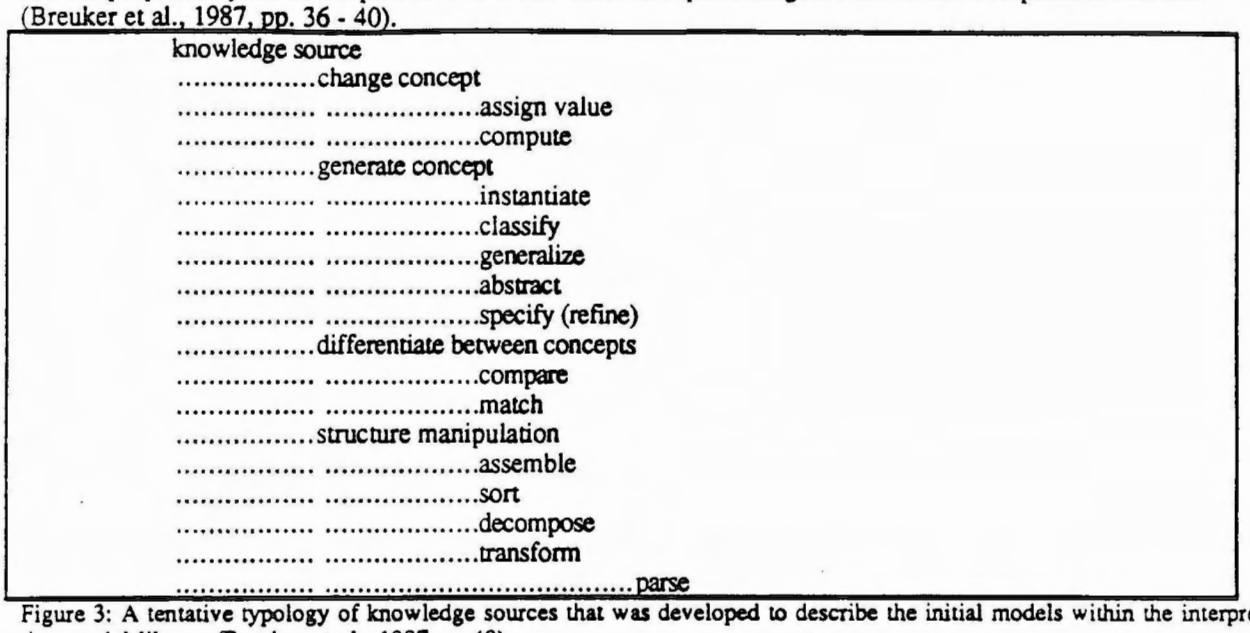


Figure 3: A tentative typology of knowledge sources that was developed to describe the initial models within the interpretation-model library (Breuker et al., 1987, p. 40).

Meta-Class	Examples	Role
Available-Observation	Karnofsky status, white-blood-cell count, platelet count	Potential inputs that can be used to assess a patient's reaction to the treatment
Available-Action	VP-16, Adriamycin, methotrexate	Medications that can be administered to the patient.
Current-Phase	VAM	The chemotherapy that is currently being applied.

Figure 4: Meta-classes for a planning task illustrated with concepts from the oncology domain. Additionally, the roles that these meta-classes play in the context of the oncology task are described.

Meta-classes describe the roles that domain-layer concepts can play in a problem-solving process. Figure 4 shows meta-classes for a planning task. It illustrates those meta-classes with concepts from the oncology domain, and explains the role of the meta-classes in the ONCOCIN problem-solving context. In the domain of cancer therapy the drug *VP-16* can be the described by the meta-class *Available-Action* (i.e., VP-16 is one of the actions that the problem solver can execute), the *white-blood-cell count* can be described by the meta-class *Available-Observation* (i.e., the outcome of the *white blood cell count* is an input that the problem solver can use). The knowledge sources and meta-classes are formulated in domain-independent problem-solving terms. For the modelling of K-ONCOCIN, new meta-classes and knowledge sources had to be created to account for the synthesis and temporal-reasoning aspects of the ONCOCIN task.

Figure 5 shows an inference structure for a simple type of time-dependent planning execution.

3.2.3. The Task Layer

At the task layer, the model describes the task decomposition that is required to control the inferences. This control structure thus results in a sequence of calls to knowledge sources defined at the inference layer. The task decomposition can be represented as a fixed hierarchy of goals that must be fulfilled through calls to knowledge sources. The task decomposition also can be represented by more complex goal structures, where the invocation of the knowledge sources depends on the results of other knowledge sources.

For the inference structure described in Figure 5, the task layer of the model might use the task decomposition described in Figure 6.

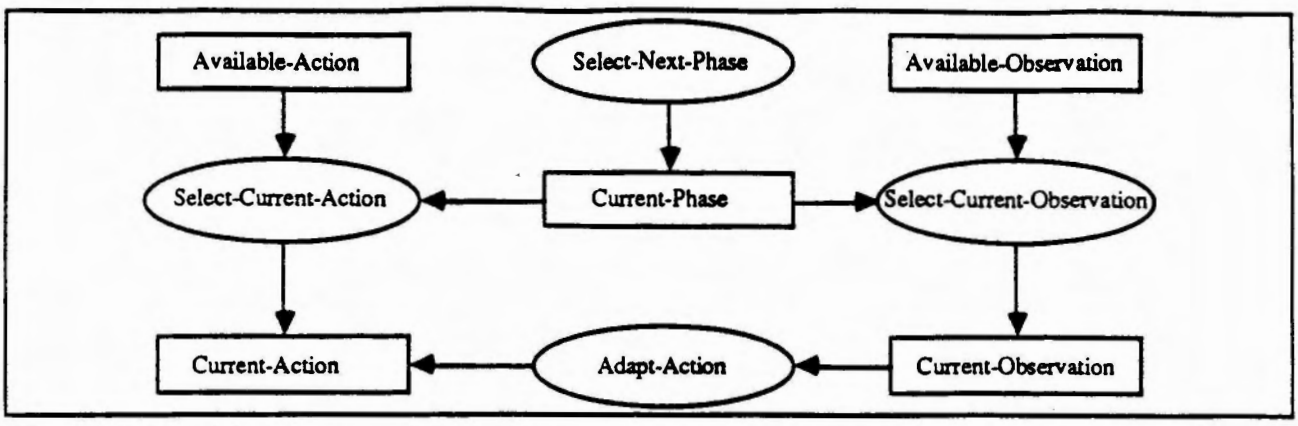


Figure 5: A simple inference structure for a simple type of time-dependent plan execution. The knowledge source *Select-Next-Phase* modifies the meta-class *Current-Phase*. The knowledge source *Select-Current-Observation* modifies the meta-class *Current-Observation* based on the meta-classes *Available-Observation* and *Current-Phase*. The knowledge source *Select-Current-Action* modifies *Current-Action* and uses *Current-Phase* and *Available-Action*. The knowledge source *Adapt-Action* modifies *Current-Action* based on *Current-Observation*.

3.2.4. The Strategy Layer

The task layer of the model represents the result of applying a single control strategy, thus represents one method with which a problem solver can use the knowledge sources and meta-classes to perform the task. A powerful problem solver can result from the task-layer description, but the control structure described at the task layer cannot allow for flexibility in problem solving. The task-layer description will cause the problem solver always to ask the same questions and cannot allow the problem solver to improve its performance, thus allowing the problem solver to fail repeatedly for the same reasons. A dynamic planner that, if necessary, could reconstruct the task layer of the model could provide the missing flexibility. The strategy-layer knowledge of a model consequently represents the design decisions and the design criteria for the task layer of the model. This knowledge could be used by a dynamic planner to build new task layers.

In our domain of protocol-based care for cancer therapy, the task layer of the model represents a skeletal-plan–refinement problem-solving method, as the general sequence of actions to take is presumed to be known to the problem solver at all times (Tu et al., 1989). If this assumption were shown to be wrong, then the dynamic planner would have to reorganize the task layer to account for the new situation with a different planning approach (e.g., hierarchical planning). To do so, the planner would use the strategy knowledge of the model.

The authors are not aware of any systems that actively use this kind of strategic knowledge.

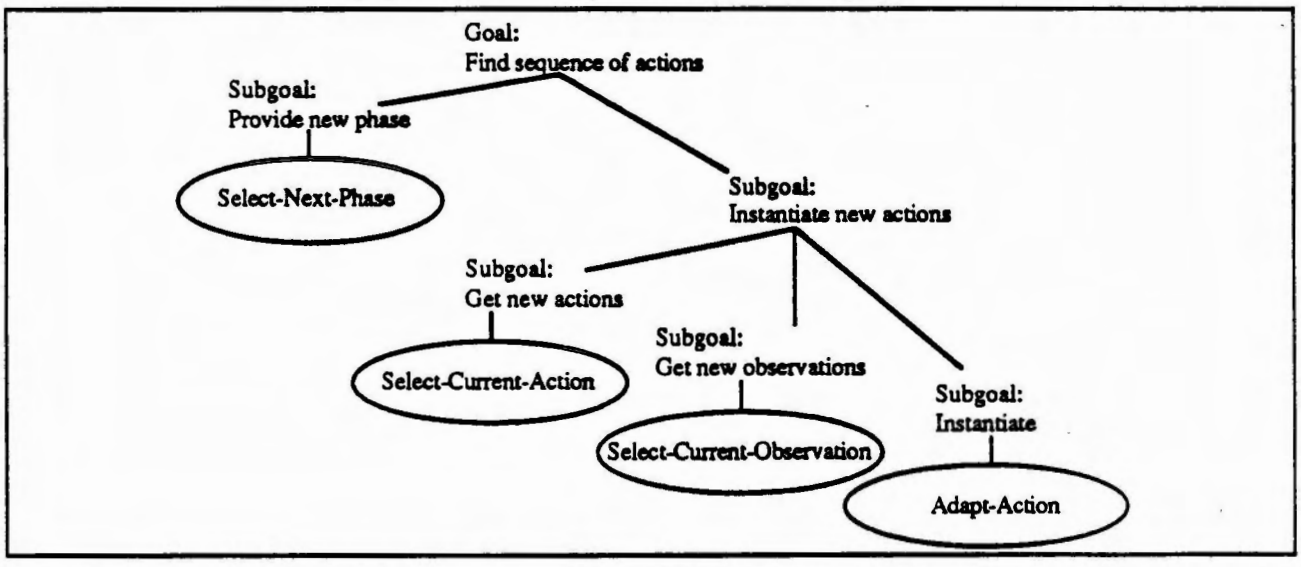


Figure 6: A task structure for the inference structure of Figure 5. To obtain the goal *Find Sequence of Actions*, the sub-goals must be pursued. The leaves of the decomposition tree correspond directly to knowledge sources that can be invoked to achieve the relevant sub-goals.

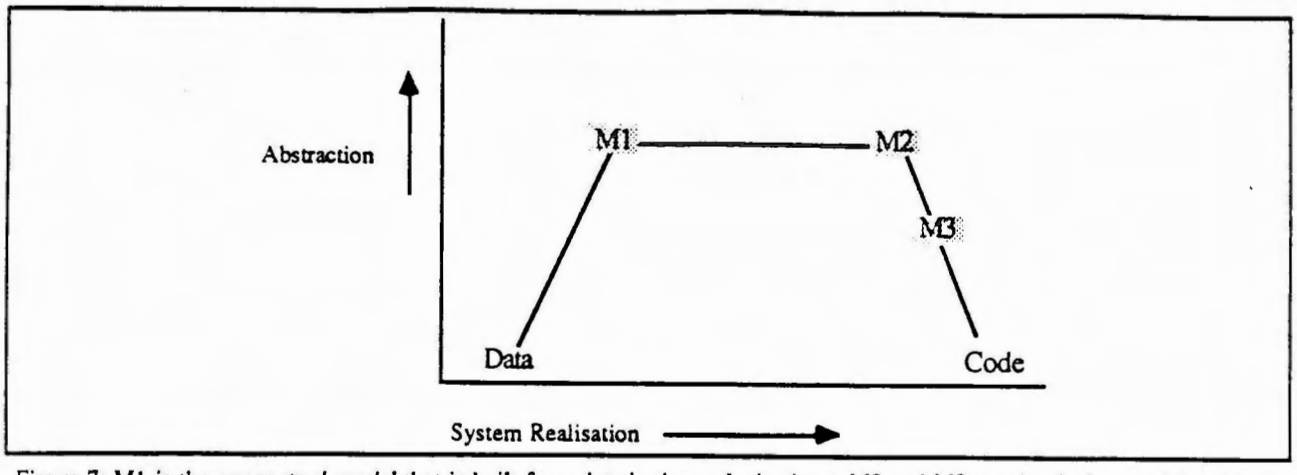


Figure 7: M1 is the *conceptual model* that is built from data in the analysis phase. M2 and M3 are the *design model* and the *detailed design models* (respectively) that ultimately result in operational program code (Breuker & Wielinga, 1989, p. 268).

3.3. The Suite of Models in KADS

The KADS methodology provides for a suite of models that lead from raw data (results of interviews, case descriptions) to operational code (Figure 7). The four layers described in the previous section can be found in each of the models. From data that result from interviews, protocol analysis, and background literature, the knowledge engineer first builds an informal model: the conceptual model (M1 in Figure 7). This model accounts for the observed expertise using the four layers of knowledge. To facilitate the interpretation of the data, KADS provides a library of interpretation models that consist of a task and an inference layer. These interpretation models serve as templates in the development of the conceptual model. The conceptual model is transformed into a design model (Bredeweg et al., 1989) that specifies the high-level programming techniques that will be used to implement the elements of the conceptual model (M2). At the same time, external requirements like modularity and speed influence the design model (M3). The design model can then be transformed by way of the detailed design model into operational code.

```
system analysis
        identify
                classify
                        simple classify
                        diagnosis
                                single fault diagnosis
                                        heuristic classification
                                        causal tracing
                                        localization
                                multiple fault diagnosis
                        assessment
                monitor
        predict
                prediction of behavior
                prediction of values
system modification
        ...
system synthesis
        transformation
        design
                transformational design
                ...
        planning
        modelling
```

Figure 8: A part of the initial library of interpretation models in KADS (Breuker et al., 1987, p. 141).

3.3.1. The Interpretation Model

A KADS interpretation model provides a domain-independent framework to help the knowledge engineer with the analysis, interpretation, and model formation in a new, not yet fully understood problem solving situation. A hierarchically organized library of interpretation models facilitates the selection of a model (Figure 8). The interpretation models of the library can be modified and adapted to the problem at hand. In case no model could be selected, the knowledge engineer builds one from scratch. Experience shows that this happens quite frequently, especially for application tasks that must be solved using constructive problem-solving methods.

An interpretation model consists of the three upper layers of KADS (i.e., the inference, task, and possibly the strategy layers). An interpretation model is a domain independent, abstract description of a problem-solving method. It gives the knowledge engineer strong indications what domain knowledge is needed as the meta-classes and knowledge sources indicated in the interpretation model must have domain-specific equivalents.

The inference layer of Figure 5, combined with the task layer of Figure 6 forms an interpretation model for a certain type of time-dependent plan execution. If the knowledge engineer chooses this interpretation model to guide acquisition of domain knowledge in a new situation, then she must look for the domain-specific equivalents of the knowledge sources and meta-classes in her analysis of verbal protocols or interviews.

An interpretation model does not prescribe a representation formalism for the domain layer knowledge. Issues of model implementation are not addressed until the knowledge engineer begins to create a design model (M2 in Figure 7).

3.3.2. The Conceptual Model

Once an interpretation model has been selected, it guides the knowledge engineer in the knowledge acquisition process by providing structure and guidance. The acquired domain knowledge, together with the interpretation model, is called a *conceptual model*. In Figure 7, the conceptual model corresponds to M1. A conceptual model is an informal notation of how a problem is to be solved (described in the upper three knowledge-modeling layers that make up the interpretation model) together with the domain knowledge that is used to do so.

A conceptual model is not operational, and cannot be executed by a computer. It represents the knowledge engineer's view on which discrete goals the system must pursue, using the underlying domain knowledge to solve a problem. The conceptual model abstracts from all details about how the problem solver will proceed. This model is meant to be a basis for documentation and for discussion among the developers about the behaviors of the future system.

3.3.3. The Design Model

The conceptual model is transformed into a set of appropriate AI techniques (e.g., A*, forward chaining, horn-clause unification) and representations (e.g., frames, rules, clauses, constraints, semantic nets) that implement the behavior that was intended by the knowledge engineer when she developed the conceptual model. In the process of transforming an abstract conceptual model into a design model, external requirements such as speed and integration are considered. In Figure 7, the design model corresponds to M2.

3.4. The Interaction of the Models

Figure 9 shows how the observed problem-solving behavior is interpreted and represented as a conceptual model, which is then transformed into a design model, which ultimately is transformed into an operational system.

The transformation between the models is a manual process that usually is done by the same knowledge engineer who developed the conceptual model. Experience with the STATCONS system (de Greef, 1989), however, has shown that the conceptual model and the design model can be developed by separate teams.

The transformation process from observed behavior via the KADS models to an operational program requires repeated interpretation and revision. Observed phenomena in verbal protocols are first captured in the conceptual model. This process necessarily introduces the knowledge engineer's bias. More important, the conceptual model is neither formal nor unambiguous. The design process thus must translate the conceptual model into unambiguous code. This part of the transformation may eliminate some interpretations and incorporate others.

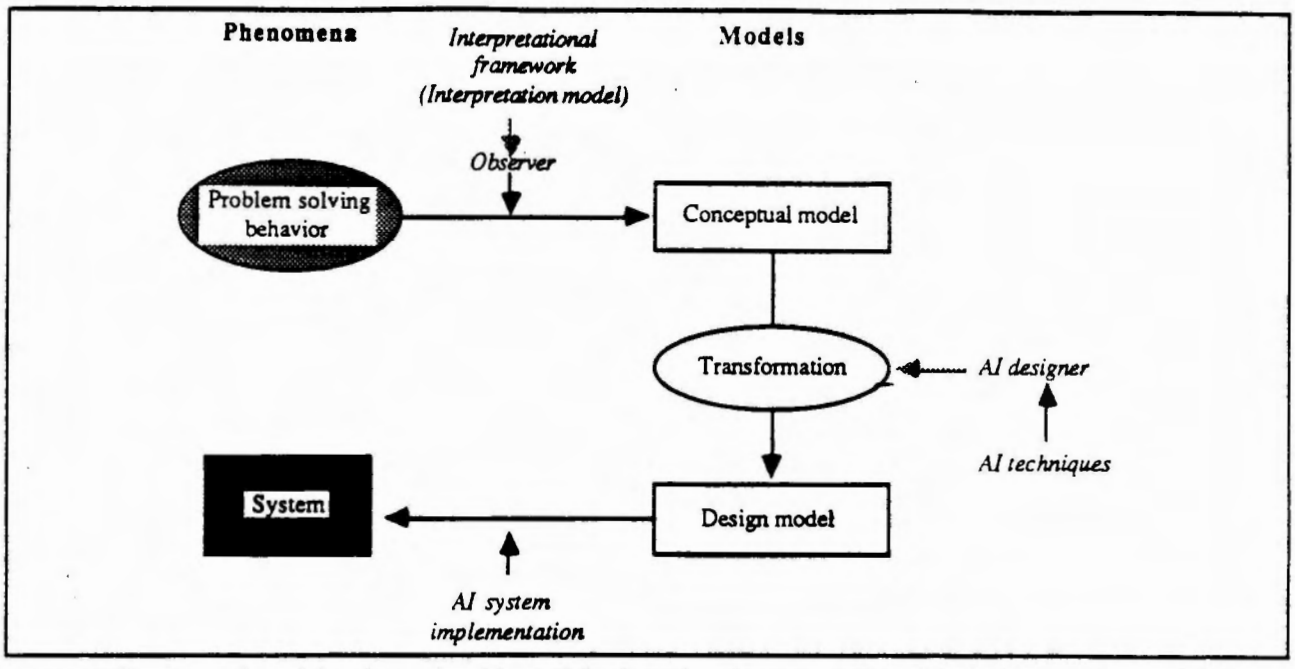


Figure 9: The interaction of the observed problem-solving behavior, the conceptual model, the interpretation model, the design model, the AI techniques, and the operational system in the KADS framework (Akkermanns & Wielinga, 1990, p. 30-11).

4. THE KADS MODELS OF THE ONCOCIN TASK

In this section we describe K-ONCOCIN—the re-modelling of ONCOCIN in KADS' terms. We will focus our description on the conceptual model of K-ONCOCIN. We will briefly present the design model; a presentation of the implemented system is beyond the scope of this paper.

We use NCOG protocol 2083-1 (Carlson et al., 1986) to illustrate K-ONCOCIN (Protocol 2083-1 has served the purpose of illustrating aspects of ONCOCIN and OPAL in a number of other publications; Musen et al., 1987; Musen, 1989b).

Figure 1 shows the schema for the protocol 2083-1. This protocol describes a clinical trial that measures the effectiveness of a new, simplified chemotherapy (called CAVP) in treating small-cell cancer of the lung. The chemotherapy (which includes the administration of the drugs Cytoxan, Adriamycin, vincristine, and predisone) is compared with the standard treatment for this tumor (two alternating chemotherapies called POCC and VAM, which require the administration of seven different drugs in total). If patients should fail to respond to CAVP, the protocol specifies that those patients receive another chemotherapy called CCoM. When physicians carry out this medical experiment, their patients with small-cell lung cancer must give their consent to be assigned at random either to Arm A of the protocol (standard therapy with POCC and VAM) or to Arm B of the protocol (experimental therapy with CAVP and, if necessary, CCoM). A second experiment within Protocol 2083-1 tests the effectiveness of prophylactic cranial irradiation (PCI) in patients who show complete response (CR) to the their assigned drug treatments (see Figure 1). Protocol 2083-1 is just one of 36 protocols currently encoded in the ONCOCIN knowledge base.

It is important to note that we limit K-ONCOCIN to that part of ONCOCIN that is accessed and used by OPAL-generated knowledge bases. This restriction means that some aspects of ONCOCIN will not be part of K-ONCOCIN.

4.1. The Conceptual Model

This section describes the KADS conceptual model for ONCOCIN. Based on the knowledge-level description of ONCOCIN (see Section 2), we shall develop the domain-layer elements of K-ONCOCIN. Based on both the inferences of ONCOCIN and the domain-layer knowledge, we will build an inference layer and a task layer that can account for the behavior of ONCOCIN in recommending cancer chemotherapy.

4.1.1. The Domain Layer

KADS requires a declarative representation of the domain-layer knowledge in terms of concepts, relations, and structures. KADS does not include special representation formalisms, such as transition nets for temporal relations, PROLOG clauses for assertions, or object structures for factual knowledge. Within the very general framework of concepts, relations, and structures, the knowledge engineer thus has considerable freedom—and very little guidance—when he defines the elements of the domain layer.

We choose an object-oriented approach for the representation of domain knowledge. For the representation of the domain-layer concepts we use hierarchical definitions of entities with attributes. Relations among entities model knowledge about how the concepts pertain to each other. Our choice of representation formalisms is strongly influenced by our experience with object-oriented knowledge representations as used in PROTÉGÉ, ONCOCIN, and BABYLON (Musen, 1989b; Guesgen, Junker & Voß, 1987; Christaller, Di Primio & Voß, 1989). Users of KADS with different backgrounds (such as PROLOG or OPS-5 programmers) probably will choose to use totally different representations.

For the specification of the temporal iteration of the therapies (the high-level algorithm of a treatment protocol shown in Figure 1), we choose to use a transition-network representation. The specification includes a *successor* relation, which is defined on unequivocal pointers to the therapies (i.e., the radiotherapies and the chemotherapies) and on the conditions that allow for a transition between therapies (e.g., random assignment of a patient to a particular arm of the protocol or progression of cancer that requires a change in chemotherapy). Figure 10 illustrates the successor relation.

We represent therapies as concepts that have attributes that define additional knowledge about the therapies, such as the therapy's duration, the drugs that are administered, laboratory tests required, maximum number of delays, and so on. For example, in Protocol 2083-1, the chemotherapy *VAM* lasts 21 days; the drugs *VP-16*, *Adriamycin* and *methotrexate* are administered as parts of *VAM*, and 2 delays of 7 days each are tolerable in the case that a patient has an unfavorable response to the treatment. Figure 11 shows the hierarchical definition of the concepts in the domain layer of K-ONCOCIN.

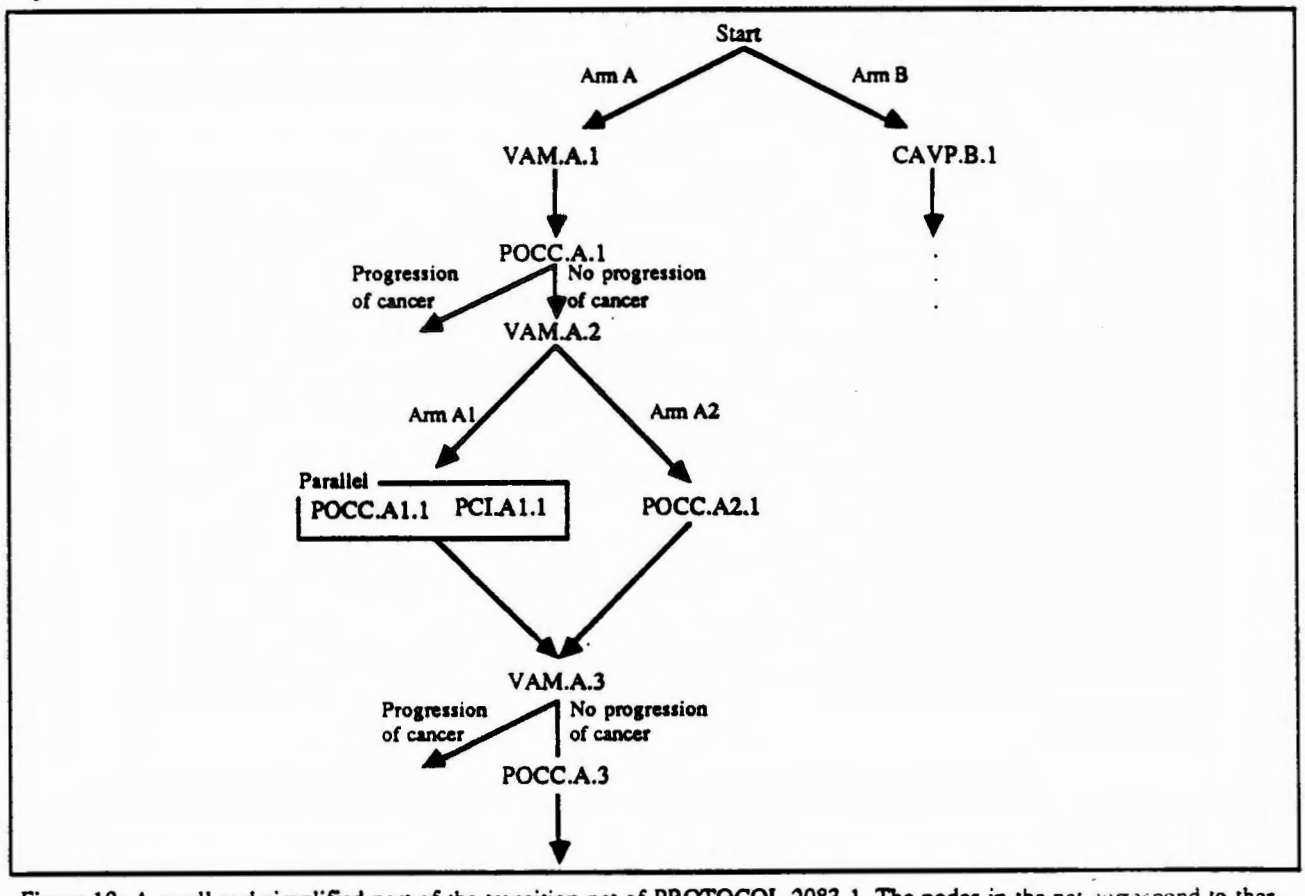


Figure 10: A small and simplified part of the transition net of PROTOCOL-2083-1. The nodes in the net correspond to therapy applications, such as VAM.A.1. VAM.A.1 stands for the first occurrence of the chemotherapy VAM in Arm A of the protocol. The node *Start* is a dummy node. The arrows between therapies represent the relation successor

For each prescription, we define attributes describing on which day in which therapy that prescription is used. Similar knowledge is represented for the laboratory tests that evaluate how the patient responds to the therapy.

In KADS, knowledge that pertains to several concepts can be represented either as attributes of concepts (i.e., binary relations) or explicitly as relations between concepts. KADS does not give any directives how to choose an attribute-based or a relation-based representation.

To develop K-ONCOCIN, we represented the following knowledge as relations between concepts:

Drug-in-Chemotherapy indicates the days of particular chemotherapies on which particular drugs are administrated (for example, *VP-16* is due on the first day of the chemotherapy *VAM).*

Test-Due indicates the days of a particular chemotherapy on which a particular test is warranted (for example, a *white-blood-cell count* is due on the first day of the chemotherapy *VAM).*

Drug-Dose-Attenuation relates the results of laboratory tests to the dosage of drugs. In Protocol 2083-1, for example, the dose of the drug *VP-16* in the chemotherapy *VAM* is reduced to 75% of the normal dose if the *white-blood-cell count* is below $3.5 * 10^3$. If the *white blood cell count* is above $3.5 * 10^3$ and the *platelet count* is between 100 and $150 * 10^3$, then the patient should receive 75% of the standard dose of *Cytoxan* in the chemotherapy *CAVP.*

Chemotherapy-Alteration relates the results of laboratory tests to the continuation or interruption of the current chemotherapies. For example, if the patient's *platelet count* is less than $100 * 10^3$, then the current chemotherapy is delayed. The duration of the delay is a property of the chemotherapy, just as is the maximum number of permitted delays before the chemotherapy is aborted.

Abstract-Disease-Activity relates descriptions of how the patient responds to the treatment to more abstract findings. For example, if the *disease activity* (a numerical representation of the state of the patient's tumor over time) equals 6, then the patient is not responding to treatment and the cancer continues to progress. If the value of *disease-activity* equals 1, then the patient shows complete response to the treatment and the cancer therefore is not progressing anymore.

We must emphasize that these entities and relations constitute but one way to view the domain-layer elements of K-ONCOCIN in the KADS framework. Because KADS is not prescriptive in the choice of formalisms for the domain layer, there are many other ways to model the same information.

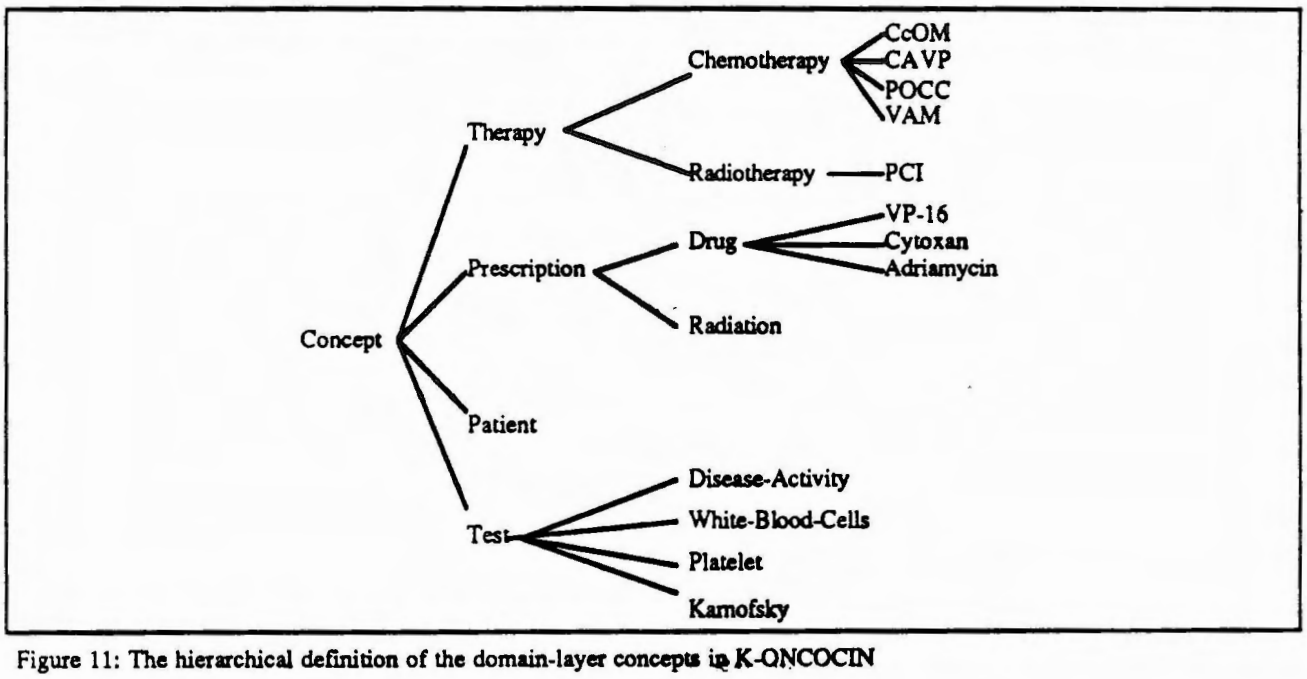


Figure 11: The hierarchical definition of the domain-layer concepts in K-ONCOCIN

4.1.2. The Inference Layer

Based on our analysis of ONCOCIN's behavior and on the domain-layer knowledge described in the previous section, we will define the elements of the inference layer for the conceptual model of K-ONCOCIN—the knowledge sources, the meta-classes, and the inference structure. The inference-layer elements are defined in domain-independent terms to

make these elements generic and thus applicable to other domains for which similar problem-solving approaches are appropriate. KADS provides only a vocabulary for the description of general inference-layer knowledge; KADS does not give strong guidance to determine precise names for the individual inference-layer elements themselves.

Many inference structures are known for analytic tasks, such as diagnosis, classification, monitoring, and prediction (see Figure 8). ONCOCIN, with its skeletal-plan–refinement problem-solving method, requires a new inference structure, which ultimately should be added to the library of known inference structures for KADS.

In the following sections, we define the knowledge sources and meta-classes of K-ONCOCIN. Before explaining how we combined these elements into an inference structure, we show how the inference-layer components are connected to the domain layer.

4.1.2.1. The Knowledge Sources

We discern the following nine classes of inferences in our model of ONCOCIN's behavior:

(1) The next state in the protocol transition net is selected based on patient data and on the current state. The state in the transition net is an unequivocal pointer to the current situation in the protocol. We account for this inference in the knowledge source *Determine-Next-State*.

(2) The current therapies are selected for the current state. As mentioned above, the current state is an unequivocal pointer to a situation in the protocol. This additional inference is required, as distinct therapies typically are administered in several different situations in the protocol. This knowledge source is called *Select-Next-Phase*.

(3) The laboratory tests that should be performed on the current day of treatment are selected by the knowledge source *Select-Current-Observation*.

(4) The interpretation of the selected laboratory tests is performed by the knowledge source *Make-Observation*.

(5) The knowledge source *Infer-Environment-Data* performs additional inferences to deduce more information from the incoming laboratory-test data, such as those described in the relation *Abstract-Disease-Activity*.

(6) The selection of the appropriate drugs to administer on the current visit is performed by the knowledge source *Select-Current-Action*.

(7) The adaptation of the drug dosage based on the results of the latest laboratory-test results and on other clinical data is attributed to the knowledge source *Adapt-Action*.

(8) The current cycle of chemotherapy is adapted to the latest results of laboratory tests by the knowledge source *Adapt-Phase*. This knowledge source decides whether the chemotherapy should be delayed, cancelled, or replaced by another drug combination if the patient shows signs of drug toxicity.

(9) The knowledge source *Select-Non-Current-Phase* determines whether the current medical action should be discontinued. For example, VAM will be discontinued if it has been active for 21 days.

4.1.2.2. The Meta-Classes

We observe several roles that domain-layer knowledge elements can play in ONCOCIN's reasoning. In K-ONCOCIN, these knowledge roles correspond to eight meta-classes.

(1) The meta-class *State* describes the currently active nodes of the protocol transition net.

(2) The meta-class *Current-Phase* specifies the therapies that are administered during the current *State*.

(3) The meta-class *Available-Phase* describes all the chemotherapies that may be relevant in a K-ONCOCIN treatment protocol.

(4) The meta-class *Current-Observation* specifies the laboratory tests that must be performed on the current day of the current therapy (described by the meta-class *Current-Phase*).

(5) The meta-class *Available-Observation* is the descriptor for all the laboratory tests that are known to K-ONCOCIN.

(6) The meta-class *Environment-Data* identifies all the data that relate to the patient under treatment protocol.

(7) The meta-class *Available-Action* points out which drugs and radiation therapies are represented in the K-ONCOCIN domain-layer knowledge.

(8) The meta-class *Current-Action* represents the drugs or radiation prescriptions that have been selected for the current day of the current therapy (described by the meta-class *Current-Phase*).

4.1.2.3. Linking Inference Layer Elements and Domain Layer Knowledge

Tables 1 and 2 show the domain-layer elements used by each knowledge source, and the domain-layer concepts that can be described by specific meta-class at certain points in the problem-solving process.

Knowledge Sources	Underlying Domain-Layer Knowledge for Protocol 2083-1
Determine-Next-State	Successor (a relation) Example: State VAM.A.1 is succeeded by state POCC.A.1
Select-Next-Phase	Relate-Protocol-State-to-Therapy (a relation) Example: VAM is the corresponding therapy for VAM.A.1
Select-Non-Current-Phase	Maximum-Nbr-Delays, Days-in-Therapy (attributes of the concept Therapy) Example: VAM lasts 21 days and can have a maximum of 2 delays
Select-Current-Action	Drug-in-Chemotherapy (a relation) Example: VP-16 is administered on the first day of VAM
Adapt-Action	Drug-Dose-Attenuation (a relation) Example: The dose of VP-16 is reduced to 75% if the WBC $< 3.5 * 10^3$
Infer-Environment-Data	Abstract-Disease-Activity (a relation) Example: If the disease activity = 6, then the patient does not respond to the therapy
Adapt-Phase	Chemotherapy-Alteration (a relation) Example: if the platelet count is less then $100 * 10^3$, then the current therapy should be delayed.
Make-Observation	Evaluate-Test (a relation)
Select-Current-Observation	Test-Due (a relation) Example: a WBC count is due on the first day of the chemotherapy VAM

Table 1: The correspondence between inference-layer knowledge sources and domain-layer knowledge.

Meta-Classes	Underlying Domain-Layer Knowledge for Protocol 2083-1
State	Protocol-State (a concept) Example: VAM.A.1 (first occurrence of VAM in arm A)
Environment-Data	Patient (a concept)
Available-Phase	Therapy (a concept) Example:VAM, PCI, POCC
Current-Phase	Therapy (a concept) Example:VAM, PCI, POCC
Available-Action	Prescription (a concept) Example: VP-16, Adriamycin, methotrexate
Current-Action	Prescription (a concept) Example: VP-16, Adriamycin, methotrexate
Available-Observation	Test (a concept) Example: Karnofsky status, white-blood-cell (WBC) count
Current-Observation	Test (a concept) Example: Karnofsky status, WBC count
Obsolete-Phase	Therapy (a concept) Example:VAM, PCI, POCC

Table 2: The correspondence between the inference-layer meta-classes and the domain-layer knowledge.

4.1.2.4. The Inference Structure

The combined meta-classes and knowledge sources form the inference structure. The inference structure can be viewed as an abstract data-dependency diagram. Knowledge sources have meta-classes as input and as output. As the inference structure of K-ONCOCIN is quite complex, we shall first present two partial structures before describing all the data-dependency relationships. The first partial structure is concerned with the selection of the appropriate laboratory tests that have to be performed on a certain day of a selected therapy. The second structure describes the selection of the proper drugs and the adaptation of their dosage.

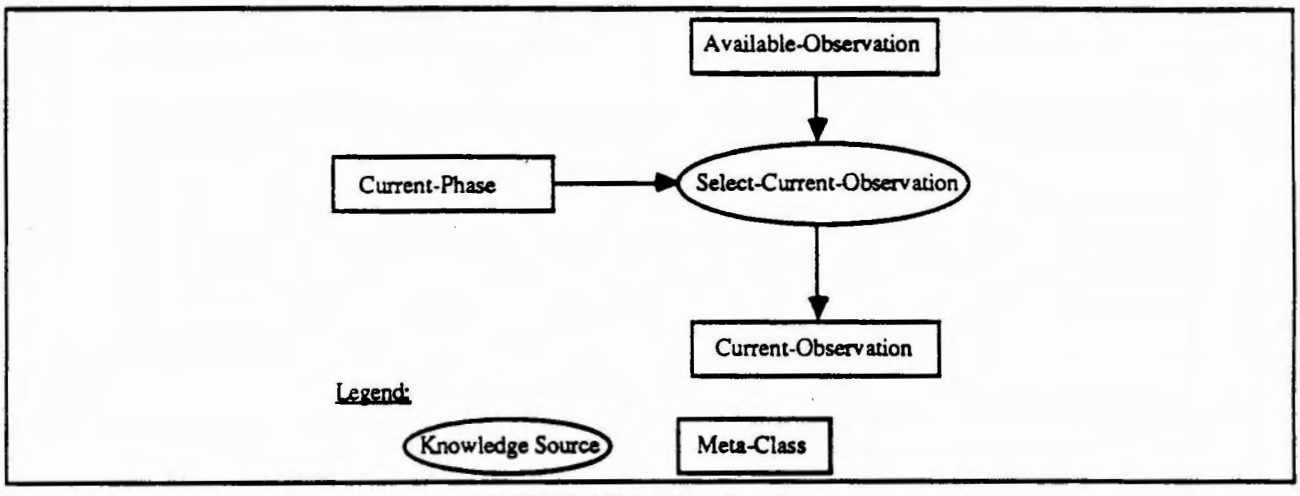


Figure 12: A part of the inference structure of K-ONCOCIN that describes the data dependency among the meta-classes *Available-Observation*, *Current-Phase*, *Current-Observation* and the knowledge source *Select-Current-Observation*. The knowledge source uses the first two meta-classes to modify the third meta-class.

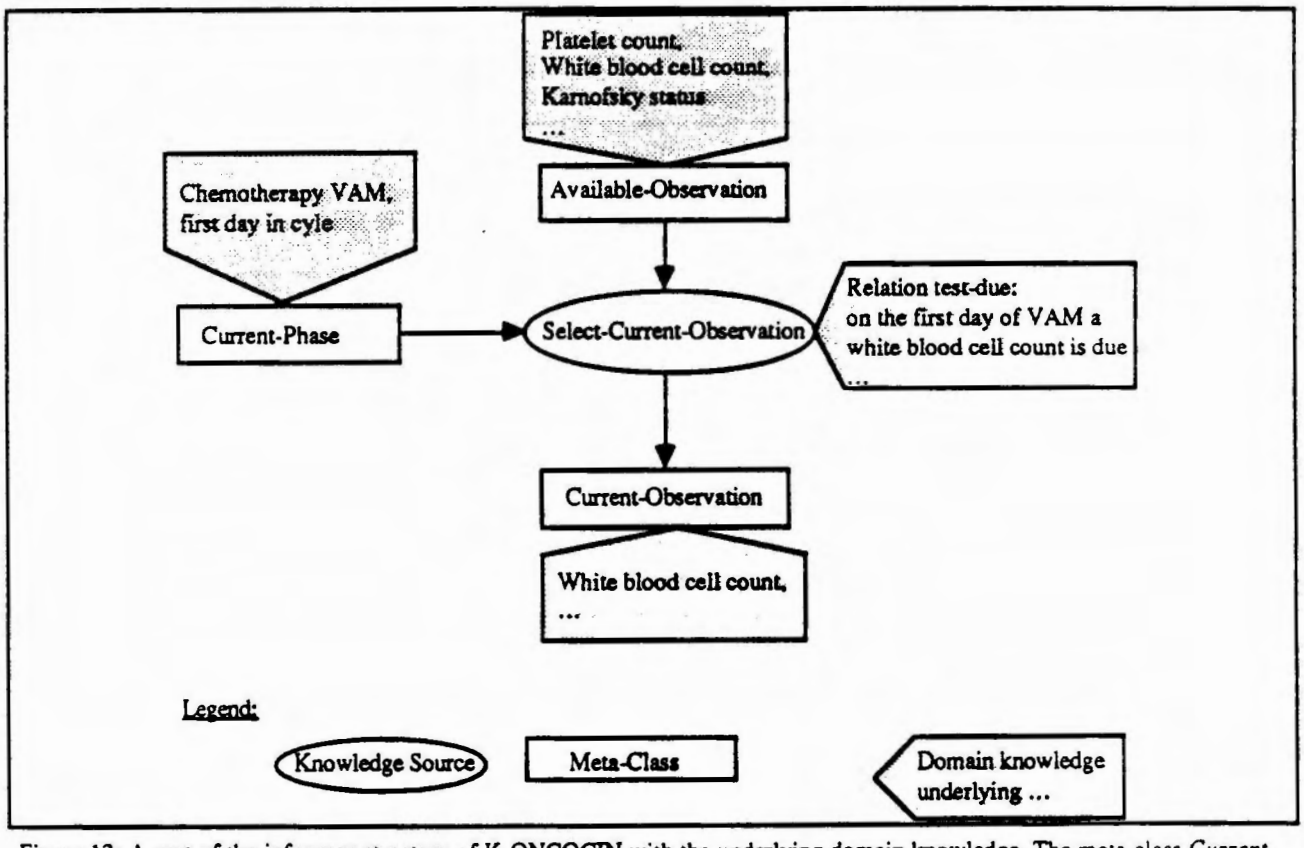


Figure 13: A part of the inference structure of K-ONCOCIN with the underlying domain knowledge. The meta-class *Current-Phase* can describe the role of the domain concept *VAM*. The meta-class *Available-Observation* models the role of the concepts *Platelet count*, *white blood cell count* and *Karnofsky status*. The knowledge source *Select-Current-Observation* uses

the knowledge of laboratory-test–ordering described in the domain relation *Test-due* to modify the meta-class *Current-Observation.*

The knowledge source *Select-Current-Observation*, which in K-ONCOCIN identifies the laboratory tests that must be performed based on the current therapy, uses knowledge about the drugs and radiation prescriptions that are available, and uses knowledge about the currently active chemotherapy or radiotherapy. The meta-classes Available-Observation and Current-Phase describe the required knowledge. The meta-class Current-Observation *describes the output* of the knowledge source. Figure 12 gives a graphical view of this data dependency.

Inference-structure elements are generic descriptions of the problem-solving roles and functions of domain-layer knowledge. Figure 13 shows some of the domain-layer elements that, at a certain point in the reasoning process, may be referenced by the inference-layer elements in Figure 12.

In Figure 14, the knowledge source *Select-Phase-Action* decides which drugs or radiation treatments (denoted by the meta-class *Available-Actions*) should be applied on the current day of the current therapy (described by the meta-class *Current-Phase*). The knowledge source *Adapt-Phase-Action* takes the selected drugs (which then correspond to the meta-class *Current-Phase-Action*) and adjusts the drugs' doses according to the patient's response to previous treatment (as reflected in the meta-class *Environment-Data*). The knowledge source *Infer-Environment-Data* uses the patient's test results (denoted by the meta-class *Environment-Data*) to derive additional abstractions that describe the patient's condition.

The dependencies between these meta-classes and knowledge sources are illustrated in Figure 14.

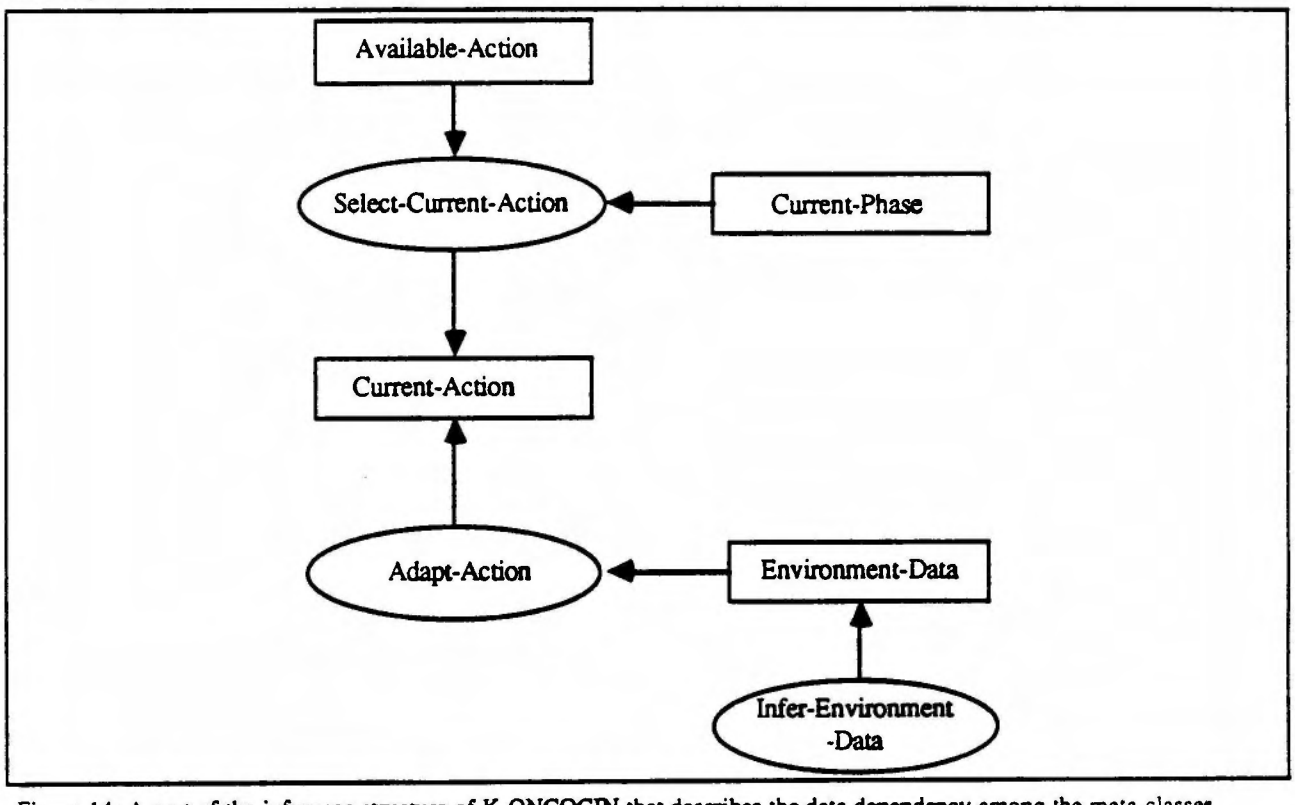


Figure 14: A part of the inference structure of K-ONCOCIN that describes the data dependency among the meta-classes *Available-Action*, *Current-Phase*, *Current-Action*, *Environment-Data* and the knowledge sources *Select-Current-Action*, *Adapt-Action* and *Infer-Environment-Data*. *Select-Current-Action* works on the meta-classes *Available-Action* and *Current-Phase*. It modifies the meta-class *Current-Action*. *Adapt-Action* modifies *Current-Action* based on *Environment-Data*. *Environment-Data* is is modified by the knowledge source *Infer-Environment-Data*.

By combining knowledge sources with their input and output meta-classes, we finally obtain the complete inference structure of K-ONCOCIN, illustrated in Figure 15. This inference structure combines the partial inference structures of Figures 12 and 14. The knowledge sources *Determine-Next-State* and *Select-Next-Phase* control the use of the transition net that describes the top-level structure of the treatment protocol (see Figure 10).

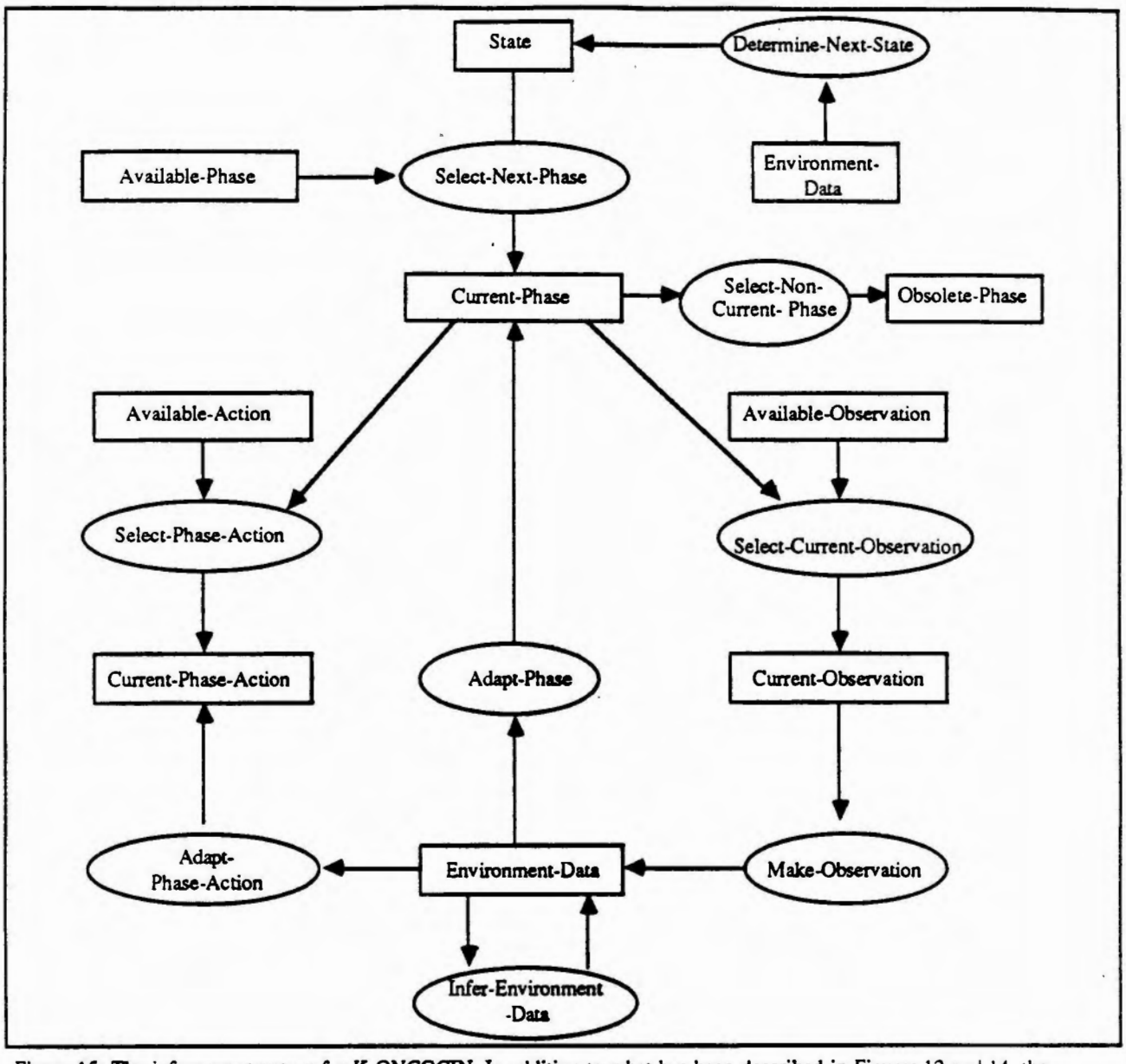


Figure 15: The inference structure for K-ONCOCIN. In addition to what has been described in Figures 12 and 14, the knowledge source *Determine-Next-State* uses the knowledge described in the meta-class *Environment-Data* to modify the meta-class *State*, which in turn is used by *Select-Next-Phase*, together with *Available-Phase*, to determine the meta-class *Current-Phase*.

4.1.3. The Task Layer

The task layer of the K-ONCOCIN model describes how the problem solver must organize its behavior as a sequence of calls to the knowledge sources. Figure 16 shows the top-level goal structure. The sub-goal *Evaluate the phase* is further decomposed as shown in Figure 17. The leaves of the goal-decomposition tree represent the knowledge sources at the inference layer that satisfy the corresponding sub-goals.

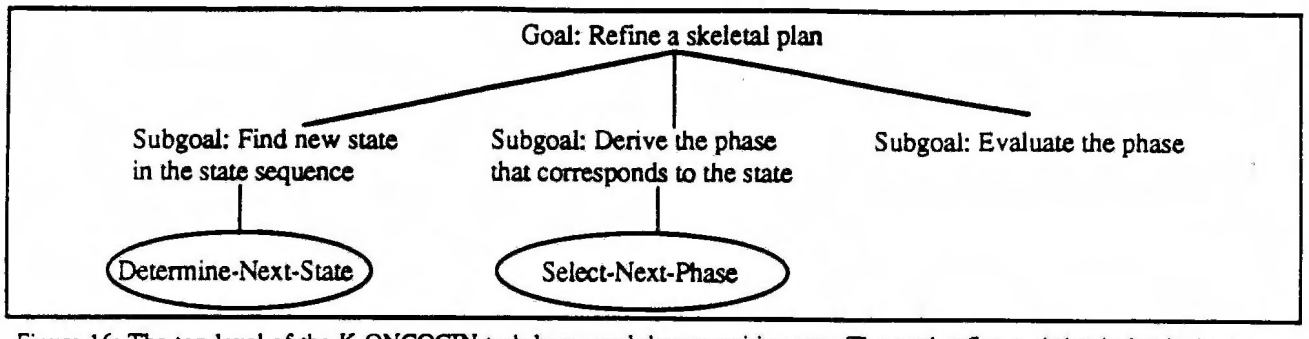


Figure 16: The top-level of the K-ONCOCIN task-layer goal decomposition tree. The goal *refine a skeletal plan* is decomposed into the sub-goals *find new state in the state sequence*, *derive the phase that corresponds to the state* and *evaluate the phase*. The first two sub-goals can be fulfilled immediately by the inference-layer knowledge sources *Next-State* and *Select-Next-Phase*. Satisfying the goal *evaluate the phase* requires further sub-goal decomposition, as shown in Figure 17.

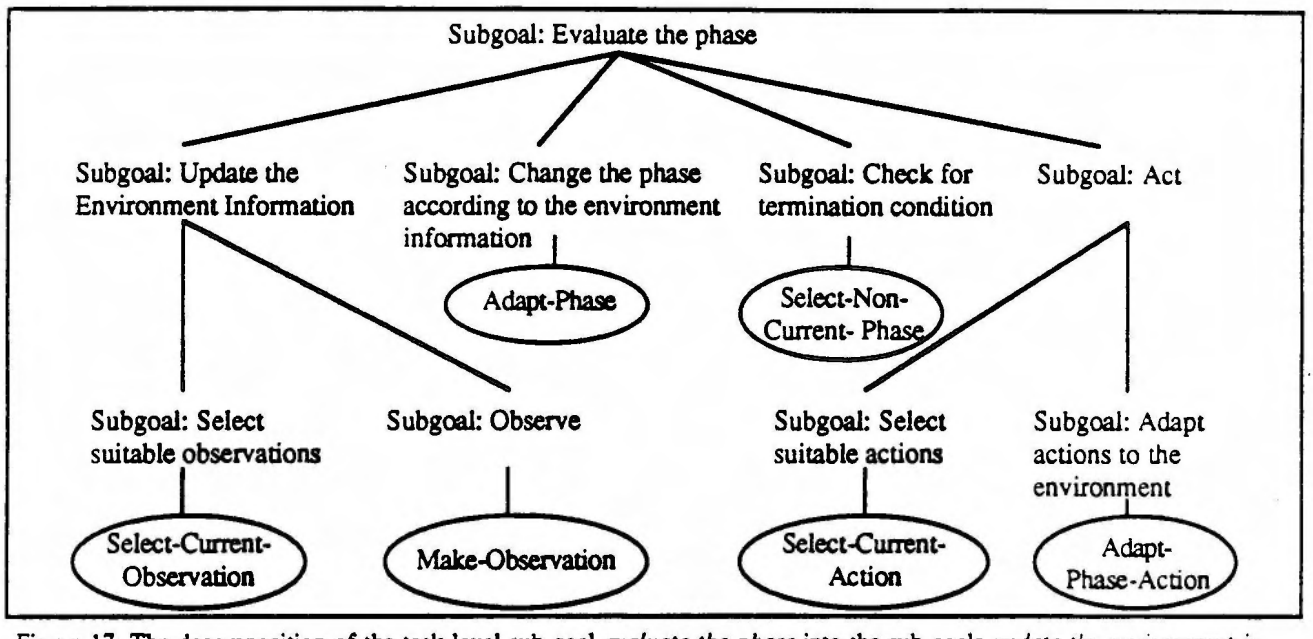


Figure 17: The decomposition of the task-level sub-goal *evaluate the phase* into the sub-goals *update the environment information*, *change the phase according to the environment information*, *check for termination condition* and *act*. Some of these sub-goals must be further decomposed before they can be fulfilled by knowledge sources at the inference level.

The K-ONCOCIN task structure is fixed; there is no need for dynamic modification of the decomposition. We therefore do not need a strategy layer to describe K-ONCOCIN.

4.2. The Design Model

In a KADS design model (see Figure 7), the knowledge engineer describes which knowledge-representation techniques will be used to implement the elements of the conceptual model. The knowledge engineer's selection of techniques necessarily will be strongly influenced by the anticipated environment (expert-system shell). In our work, we have used BABYLON V 2.1 (Di Primio & Wittur, 1987; Guesgen, Junker & Voß, 1987; Christaller, Di Primio & Voß, 1989) to implement a functioning version of K-ONCOCIN. BABYLON V 2.1 is LISP-based and integrates frames, PROLOG, rules, and constraints as knowledge-representation formalisms. K-ONCOCIN runs on a Macintosh and solves the task of cancer-chemotherapy administration for the protocol 2083-1. A detailed description of the K-ONCOCIN design model is forthcoming (Linster, 1991). The same technique was used to implement a conceptual model for clamping tool selection for lathe turning (Kühn, Linster & Schmidt, 1991).

In brief, we chose the frame and frame-instance representation mechanisms of BABYLON V 2.1 to encode concepts and attributes from the domain layer. This formalism allows for explicit preservation of the hierarchical relationships among the domain layer concepts shown in Figure 11. To implement the relations of the domain layer (e.g., Drug-in-Chemotherapy, Test-Due, and Drug-Dose-Attenuation), we used PROLOG clauses, as PROLOG allows for extensional description of relations through the enumeration of the tuples and for intentional description through characteristic predicates that are true for the tuples that satisfy the relations.

The inference-layer elements also are represented explicitly. Meta-classes are rendered as dynamic PROLOG predicates that return *true* for those domain-layer concepts that currently play the role in problem-solving indicated by the meta-class. For example, we represent the meta-class *Current-Observation* in the design model by a predicate called *Current-Observation* that is satisfied by the domain-layer concepts such as *Test*, *Disease-Activity*, and so on, while these concepts play the role denoted by the meta-class. The knowledge sources are implemented as abstract rule schemata that can be related to the underlying domain relations. At run time, the problem solver instantiates the rule schemata with the necessary concepts and relations from the domain layer to form a set of executable forward-chaining rules.

The task layer of the conceptual model is implemented as a sequence of calls to the rule schemata that implement the knowledge sources.

Developers usually consider external requirements such as speed, integration, and data management in the design model. As K-ONCOCIN was developed purely as an academic exercise, we did not develop a special interface, or a patient database.

5. DISCUSSION

We used KADS to re-model ONCOCIN as part of an experiment designed to understand the methodology better; we did not turn to KADS to solve a real problem in knowledge engineering. K-ONCOCIN was developed not only from original medical protocol documents, but also from the data structures of ONCOCIN and OPAL. These previous computational models of the task of administering cancer therapy provided us with biased preconceptions, and may have caused us to use KADS differently from when KADS is used to solve more traditional problems in knowledge engineering. Nevertheless, we believe that our experience in developing K-ONCOCIN offers us an opportunity to make useful observations regarding the KADS approach.

5.1. Guidance for Knowledge Engineering

KADS is not like a cookbook for building expert systems. In fact, KADS offers little guidance concerning how to construct a conceptual model and how to transform that conceptual model into a design model. The approach does provide an abstract ontology for describing problem solving, consisting of concepts, structures, and relations on the domain layer; knowledge sources, meta-classes, and inference structures for the inference layer; and goals and tasks for the task layer. KADS does not give directives how to use this ontology in a given knowledge-engineering situation, but it does offer a concise terminological framework for the description of problem solving. For example in the domain layer of our task, KADS does not tell us whether we should view *chemotherapies* as complex structures that combine *consists-of* and *temporal* information to express the combinations of drugs and tests that make up a chemotherapy, or if this information should be represented separately by specifying temporal relations, as we have done in the relations *Test-Due* and *Drug-in-Chemotherapy*.

By supplying typologies of meta-classes, knowledge sources (Figure 3) and inference structures (Figure 8), KADS provides a useful set of terms and relations for modeling knowledge. This ontology, however, is far from being sufficient to categorize the universe of problem-solving behavior. For example the library of interpretation models does not provide an inference structure that we could have used to model our task. The inference structure that was closest to our task is called *hierarchical design* (Breuker et al., 1987, p. 96). This structure, apart from the fact that it does not consider sequences of actions, does not provide the detailed modeling that our task required, as it contains knowledge sources such as *decompose*, *specify*, *design* and *aggregate*.

KADS does not provide guidance in the use of the ontology to categorize novel phenomena in a knowledge-engineering process. This ability would require directives for the representation of temporal, causal, factual, and all other kinds of knowledge on the domain layer. On the inference layer, such guidance would require a richer typology of inference-layer description elements and explicit criteria for the recognition of knowledge sources and meta-classes in verbal documents and transcripts. Given the current state of knowledge-acquisition research, however, the current KADS framework directs the knowledge engineer as best as is possible in the context of an open-ended modeling environment.

KADS—rather than being prescriptive—offers a structured terminology that the knowledge engineer can use to create a conceptual model that may express a wide range of problem-solving methods in virtually any application domain. The transition between the conceptual model and the design model (see Figure 9), however, requires the trained knowledge engineer to use skill in reconstructing the knowledge in the new, more restricted format. Although KADS thus offers considerable freedom in helping the developer to build a conceptual model, the technique does not assist

the developer to overcome the bias and information loss that can occur during the model-transformation phases.[2] Model transformation is still a poorly understood process that relies heavily on the knowledge engineer's intuition and programming skills. Nevertheless, making explicit the intermediate stages between the initial conceptual model and the ultimate program code can be viewed as a very valuable step toward more principled approaches to knowledge engineering.

5.2. Other Knowledge-Acquisition Approaches

Automated knowledge acquisition tools, such as SALT (Marcus, 1988; Marcus, 1989), KNACK (Klinker et al., 1989; Klinker, 1989), PROTÉGÉ/p-OPAL (Musen, 1989b) and OPAL (Musen, 1987) take a different view on the problem of knowledge engineering. Such tools provide a predefined problem-solving method, a set of generic actions and data-structures, and a specialized editor with which users can instantiate the generic actions and data structures with domain-specific knowledge (Figure 18).

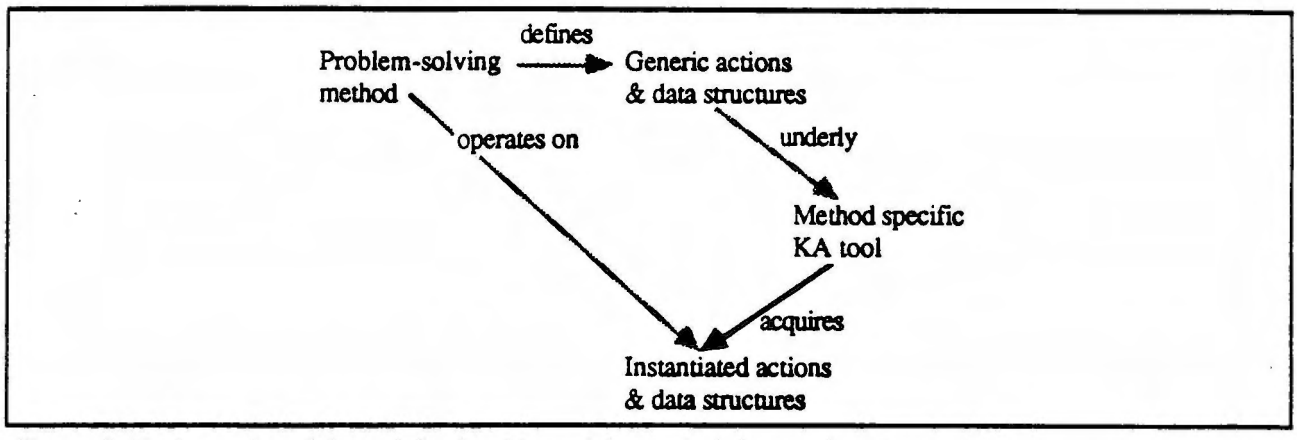


Figure 18: The interaction of the predefined problem-solving method, the generic actions and data-structures, and the specialized knowledge-acquisition editor in a SALT-like method-based tool (Karbach, Linster & Voß, 1990, p. 295).

The user thus must fit knowledge about the application to the problem-solving method and its data-structures. For example, the generic actions and data-structures in SALT correspond to the schemata for fixes, constraints, and the design extensions that are used in a problem-solving method known as *propose-and-revise* (Marcus, 1988). The knowledge that SALT can acquire must be expressed only in these terms. An extensive discussion of assumptions built into tools such as SALT has been presented elsewhere (Karbach, Linster & Voß, 1990).

If a suitable knowledge-acquisition tool is available for an application, then the use of that tool may allow for efficient knowledge entry and may provide helpful insights into the emerging knowledge base. The knowledge engineer, however, must know in advance whether a given tool is suitable for an application task at hand. KADS may be able to aid the knowledge engineer in this determination only if there exists a KADS model of the tools that can be compared with a KADS conceptual model of the application.

KADS conceptual models are application-specific, and thus contain all the knowledge needed to solve a particular task in a particular application domain. The upper layers of KADS conceptual models, however, are not application specific. The task layer and the interpretation layer are described in generic terms. These two layers constitute a description of a domain-independent problem-solving method.

The domain layer of a conceptual model can be divided into (1) the type declarations and (2) the instance definitions. This class–instance distinction applies to the relations as well, which are first declared and then instantiated with n-tuples. In the domain layer of K-ONCOCIN (see Figure 11), the type declarations include *Chemotherapy, Radiotherapy, Drug, Radiation, Patient* and *Test*. The corresponding instances include *CcOM, CAVP* and so on.

Thus, within KADS conceptual models, we can distinguish among the model of the problem solving-method (the strategy, task and interpretation layers of the model), the model of the task (the problem solving model with the type declarations from the domain layer), and the application model (the conceptual model as a whole). The KADS approach thus could serve as an initial step in the development of knowledge acquisition tools like SALT (which are

---

[2]After development of the final system, knowledge engineers must continuously revise the documentation for the conceptual model to reflect changes made directly to the program code to preserve the connection between implemented system and documentation.

based on models of problem-solving methods), tools like OPAL (which are based on models of tasks), in addition to aiding the conceptual design of complete expert-system applications.

5.3. Lessons from K-ONCOCIN

Based on our development of K-ONCOCIN, we can identify several issues that researchers should address as they work to extend and refine the KADS method.

The library of existing interpretation models must be extended to cover a wider range of problems. The elements of the library need to be described in much greater detail. To select an interpretation model, the knowledge engineer needs to refer to a similar system that was built to solve a similar task, using the approach that Bennett (Bennett, 1985) adopted in ROGET. Novice users of the KADS approach, however, need to know more than simply which interpretation models were used previously for modeling particular tasks; novices need extensive case studies that can be used as exemplars for the application of the KADS methodology. One of our goals in developing K-ONCOCIN was to document such an exemplar.

Currently, the interpretation models of the library (Figure 8) do not describe the domain layer phenomena they explain and the knowledge that they require. It is thus difficult to recognize whether a complex interpretation model, such as *planning* is applicable to a task at hand, or if the knowledge engineer should develop a new one. This description may be informal and need not be stated as data-structures. For example if a knowledge engineer wants to apply the interpretation model of K-ONCOCIN, then the task must be solvable by a sequence of actions that are executed in a predefined order, and the fine-tuning of the sequence and of the actions depends on tests or observations. These building-blocks must be present in the knowledge that was elicited. Otherwise our model is not applicable.

Currently, the knowledge engineer manually transforms the conceptual model into a design model, which she then transforms manually into an operational system. Every change that is made to the conceptual model requires transformation of the corresponding documentation and, ultimately, transformation of the operational program code. Automated tools are needed urgently, so that the details of the domain-layer knowledge can be maintained and adapted by application specialists on site after delivery of the final expert system. This requirement calls for p-OPAL like editors that allow their users to view the knowledge base in familiar terms and to change the domain knowledge within the framework defined by the KADS conceptual model.

ACKNOWLEDGMENT

We thank Werner Karbach, Otto Kühn, Karin Lagrange and Angi Voß for comments on earlier versions of this paper.

REFERENCES

Akkermanns & Wielinga 1990. Akkermanns, G.S.H. and Wielinga, B. On problems with the knowledge level perspective. In *Proceedings of the 5th knowledge acquisition for knowledge based systems workshop*, Gaines, B.R. and Boose, J.H., University of Amsterdam, University of Calgary, 1990, pp. 30/1 -30/20.

Akkermans et al. 1990. Akkermans, H., Balder, J. Frank van Harmelen, Schreiber, G. and Wielinga, B. Formal Specifications of Knowledge Level Models. REFLECT Report 1.4, Esprit Basic Resarch Project P3178, ECN Petten, 1990

Alexander et al. 1987. Alexander, J., Freyling, M., Shulman, S., Rehfuss, S. and Messick, S. Onotological analysis: an ongoing experiment. *International Journal of Man-Machine Studies 26*, 4 (1987).

Anjewierden & Wielemaker 1990. Anjewierden, A. and Wielemaker, J. Shelley - Computer aided knowledge engineering. In *Proceedings of EKAW90*, Wielinga, B., Boose, J., Gaines, B., Schreiber, G. and van Someren, M., IOS Press, Amsterdam, 1990, pp. 41 - 59.

Bennett 1985. Bennett, J.S. ROGET: A Knowledge-Based System for Acquiring the Conceptual Structure of a Diagnostic Expert System. *Journal of Automated Reasoning 1*(1985).

Boose, Shema & Bradshaw 1989. Boose, J.H., Shema, D.B. and Bradshaw, J.M. Recent Progress an AQUINAS: a Knowledge Acquisition Workbench. *Knowledge Acquisition 1*, 2 (June 1989), 185 - 214.

Bredeweg et al. 1989. Bredeweg, B., de Greef, P., Schreiber, G., Terpstra, P., Wielinga, B., Brunet, E., Simonin, N. and Wllyn, A. A KADS Approach to KBS Design. Tech. Rept. B6, KADS Report, ESPRIT Project P1098, University of Amsterdam, Amsterdam, The Netherlands, September, 1989.

Breuker et al. 1987. Breuker, J., Wielinga, B., van Someren, M., de Hoog, R., Schreiber, G., de Greef, P., Bredeweg, B., Wielemaker, J., Billaut, J.P., Davoodi, M. and Hayward, S. Model-Driven Knowledge Acquisition: Interpretation Models. Tech. Rept. Esprit Project 1098, Deliverable Task A1, University of Amsterdam, Social Sciences Informatics, Amsterdam, 1987.

Breuker & Wielinga 1989. Breuker, J. and Wielinga, B. Models of Expertise in Knowledge Acquisition. In *Topics in Expert System Design, Methodologies abd Tools*. North-Holland, Guida, G. and Tasso, C., pp. 265 - 295, Amsterdam, 1989.

Carlson et al. 1986. Carlson, R.W., Sikic, B.I., Gandara, B.R., Cadman, E.C. and Friend, N. A phase III trial of seven-drug vs. three-drug chemotherapy regimens with or without prophylactic cranial irradiation for undifferentiated small cell anaplastic lung cancer: extensive disease. Tech. Rept. 2083-1, Protocol, Northern California Oncology Group, Palo Alto, 1986.

Chandrasekaran 1988. 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.

Christaller, Di Primio & Voß 1989. Christaller, T., di Primio, F. and Voß, A. *Die KI-Werkbank BABYLON*, Addison Wesley, Bonn (1989).

Clancey 1985. Clancey, W. Heuristic Classification. *Artificial Intelligence* 27(1985), 289 - 350.

Gaines & Shaw 1990. Gaines, B.R. and Shaw, M.L.G. Cognitive and logical foundations of knowledge acquisition. In *Proceedings of the 5th knowledge acquisition for knowledge based systems workshop*, Gaines, B.R. and Boose, J.H., University of Calgary, Calgary, 1990.

De Greef et al. 1988. de Greef, P., Breuker, J., Schreiber, G. and Wielemaker, J. Stacons: Knowledge Acquisition in a Complex Domain. In *Proceedings of ECAI88*, Kodratoff, Y., ECCAI, Pitman, Munich, 1988, pp. 100 - 105.

Gruber 1989. Gruber, T.R. A Method for Acquiring Strategic Knowledge. *Knowledge Acquisition 1*, 3 (September 1989), 255 - 277.

Guesgen, Junker & Voß 1987. Guesgen, H.W., Junker, U. and Voß, A. Constraints in a Hybrid Knowledge Representation System. In *Proceedings of IJCAI87*, McDermott, J., IJCAI, Morgan Kaufmann, Los Altos, 1987, pp. 30 - 33.

Karbach, Linster & Voß 1990. Karbach, W., Linster, M. and Voß, A. Models of Problem Solving in Knowledge Based Systems: Their Focus on Knowledge Acquisition. *Knowledge Acquisition* (December 1990).

Klinker et al. 1989. Klinker, G., Boyd, C., Dong, D., Maiman, J., McDermott, J. and Schnellbach, R. Building Expert Systems with KNACK. *Knowledge Acquisition 1*, 3 (September 1989), 299 - 320.

Klinker 1989. Klinker, G. A Framework for Knowledge Acquisition. In *Proceedings of EKAW89*, Boose, J.H., Gaines, B.R. and Ganascia, J.G., Universite Paris-Sud, 1989, pp. 102-116.

Kühn, Linster & Schmidt 1991. Kühn, O., Linster, M. and Schmidt, G. Clamping, COKAM, KADS and OMOS, submitted for EKAW91, Crieff, 1991

Linster 1991. Linster, M. Implementing K-ONCOCIN with OMOS, Technical report of GMD, Forthcoming, 1991

Marcus 1988. Marcus, S. A Knowledge-Acquisition Tool for Propose-and-Revise Systems. In *Automating Knowledge Acquisition for Expert Systems*. Kluwer Academic Publishers, Marcus, S., Ch. 4, pp. 81 - 124, Boston, 1988.

Marcus 1989. Marcus, S. Understanding Decision Ordering from a Piece-Meal Collection of Knowledge. *Knowledge Acquisition 1*, 3 (September 1989), 279 - 298.

McDermott 1988. McDermott, J. Preliminary Steps Toward a Taxonomy of Problem-Solving Methods. In *Automating Knowledge Acquisition for Expert Systems*. Kluwer Academic, Marcus, S., Ch. 8, pp. 225 - 256, Boston, 1988.

Musen et al. 1987. Musen, M.A., Fagan, L.M., Combs, D.M. and Shortliffe, E.H. Use of a domain-model to drive an interactive knowledge-editing tool. *International Journal of Man-Machine Studies* 26(1987), 105 - 121.

Musen 1989a. Musen, M.A. Automated Support for Building and Extending Expert Models. *Machine Learning, Special Issue on Knowledge Acquisition* 4, 3, 4 (1989), 347 - 376.

Musen 1989b. Musen, M.A. *Automated Generation of Model-Based Knowledge Acquisition Tools*, Pitman Publishing, London , Research Notes in Artificial Intelligence(1989).

Musen 1989c. Musen, M.A. *Conceptual Models of Interactive Knowledge Acquisition Tools*, Knowledge Acquisition 1 (1989), 73 - 88

Newell 1982. Newell, A. The Knowledge Level. *Artificial Intelligence* 2(1982), 87 - 127.

Di Primio & Wittur 1987. di Primio, F. and Wittur, K. BABYLON, a Meta-Interpretation Model for Handling Mixed Knowledge Representations. In *Proceedings of the 7th International Workshop on Expert Systems and Their Applications*, Avignon, 1987, pp. 821 - 833.

Punch & Chandrasekaran 1990. Punch, W.F. and Chandrasekaran, B. An Investigation of the Roles of Problem-Solving Methods in Diagnosis. In *Proceedings of the Tenth International Workshop: Expert Systems and their Applications*, EC2, Avignon, France, 1990, pp. 25 - 36.

Schachter-Radig & Wermser 1988. Schachter-Radig, M.J. and Wermser, D. A Sales Assistant for Chemical Measurement Equipment. In *Proceedings of ECAI88*, Pitmann Publishing, 1988, pp. 191-193.

Schreiber et al. 1988. Schreiber, G., Breuker, J., Bredeweg, B. and Wielinga, B. Modelling in KBS Developement. In *Proceedings of EKAW88*, Boose, J.H., Gaines, B.R. and Linster, M., GMD, St. Augustin, 1988, pp. 7/1 - 7/15.

Shaw & Gaines 1987. Shaw, M.L.G. and Gaines, B.R. KITTEN: Knowledge initiation and transfer tools for experts and novices. *International Journal of Man-Machine Studies* 27(1987), 251 - 280.

Shortliffe et al. 1981. Shortliffe, E., Scott, A., Bischoff, M. and van Melleand C. Jacobs, W. ONCOCIN: An expert system for oncology protocol management. In *Proceedings of the 7. International Joint Conference Artificial Intelligence*, 1981, pp. 876-881.

Tu et al. 1989. Tu, S.W., Kahn, M.G., Musen, M.A., Ferguson, J.C., Shortliffe, E.H. and Fagan, L.M. Episodic Skeletal-Plan Refinement Based on Temporal Data. *Communications of the ACM 32*, 12 (December 1989), 1439 - 1455.

Voß et al. 1990. Voß, A., Karbach, W., Drouven, U. and Lorek, D. Competence assessment in configuration tasks. In *Proceedings of ECAI90*, Aiello, L.C., ECCAI, Pitman, Stockholm, This paper got the DIGITAL European AI Research Paper Award , 1990, pp. 676 - 681.

Wielinga & Breuker 1986. Wielinga, B. and Breuker, J. Models of expertise. In *European Conference on AI*, Brighton, 1986.

Wielinga et al. 1989. Wielinga, B., Akkermanns, H., Schreiber, G. and Balder, J. A knowledge acquisition perspective on knowledge level models. In *Proceedings of the 5th knowledge acquisition for knowledge based systems workshop*, Boose, J.H. and , B.R.G., University of Calgary, Calgary, 1989.

Most emphasis will be on life cycle models (LCM), executable specifications (ExSpec), and prototypes.[3] Life cycle approaches have been selected because they seem to be the most spread ones in the daily practice of SE and because the strongly discussed KADS KEngg method can also be discussed under this paradigm. Executable specifications receive high research attention in software engineering from theoretical computer science institutions and are also characteristic for one branch of KADS research. Prototyping as kind of a tentative counterpart to the more systematic other approaches shall also play some role in the subsequent text.

2.4 Criteria Applied to the Approaches

If applicable, these approaches will be discussed w.r.t four criteria. In both, SE and KEngg, the early phases are regarded as the most critical ones because of their inevitable contamination with real world informality ([Itzfe89; Twine89]). Hence the first three out of four criteria applied in this comparison put emphasis on aspects of the early phase. I.e. we shall first analyze, what is the *complexity* and the *perspective* of the languages recommended or prescribed for the results of the first analytic activities and what *further guidance* is provided *for analysis*. Subsequently we shall investigate, how *further steps* towards code are performed, supported etc.

2.4.1 Complexity of the First Language

Both SE and KEngg deal with complex phenomena in the world and with complex software products as their final outcome. Though differences in complexity can be assumed, principally any complex task needs adequate linguistic means to express its details. Inadequate languages hinder or may even prevent assessment of a phenomenon (cf. [Heise89]). Therefore, analysis of the languages, which the approaches have selected or created for there first assessment of an area of application allows to draw some conclusions about their scope and limitations.

2.4.2 Perspective of the First Language

Possible perspectives for a software product to be started with are its hardware environment, its internal behaviour, its behaviour relative to the outside world etc. These starting points are qualitatively different and differently far from the final product. Being far from the final product and its constraints may on one hand allow more freedom for the first notation but may make the subsequent tasks more difficult. I.e. the perspective is an additional important trait of the first notation to assess the scope and limitations of a method which it represents.

2.4.3 Analytic Guidance

Analytic guidance is more than a language in which to denote ones conceptualizations of an area of application. Languages can be better or worse suited for expressing certain phenomena, but analytic guidance furthermore means to make suggestions or prescriptions how to map something in the world or a trait of the to be built software product into expressions of a language. In its strongest sense it would mean to identify referents in the world for terms in a theory, as for instance physical objects taken as referents for concepts such as ideal gas, momentum etc..

---

[3] still under work; can be delivered for the full final paper

The analogy indicates that a theory or model of the to be mapped area of application or class of applications suggests itself as a mental tool for the SE or KEngr to analyze and make sense of the possibly scattered pieces of information he receives when investigating requirements, expertise etc.. On the other hand, when outlining the variety of approaches we had claimed that theories don't exist except for very limited domains. Hence we cannot but accept that guidance is based on more or less consolidated assumptions instead of mature theories.

2.4.4 From First Notation to Code - The First Step

Before going into detail, we want to justify the use of "First Notation" (as we have used "first language" above). The term is meant to subsume specification, requirements, conceptual model, etc. according to the individual terminology of a method under investigation. Whatever the first document that not (only) establishes the organizational or environmental constraints of the future product but deals with its structure, behaviour etc. is taken as the starting point.

Though there seems to be agreement that the primary assessment and hence the first notation of an area of application is the most critical one in the whole development of software, the subsequent steps are absolutely not trivial. Although in some way the first notation attempts (for good reason) to abstract from implementation detail, implementation has to be achieved in the end. I.e. we have to identify sequences of activities respectively intermediate results (see above, [Boehm88])., which can be carried out respectively delivered in a principled way. Essential distinctions concerning the first step are whether

- the first notation "fully specif(ies) a system functionally without fully specifying its implementation" (as [Balze78] puts it)
- formal or informal first notations help to this end [Twine89]
- the first notation is domain directed or implementation directed (see above).

3.0 Life Cycle Models (LCM)

"The primary functions of a software process model are to determine the *order of stages* involved in software development ... and to establish the *transition criteria* for progressing ... " ([Boehm88], p.61). This has to be distinguished from individual methods how to achieve and document the stages, although one can hardly be discussed without the other. Fine grain differentiations between life cycle models hence concern the two question according to which criteria the sequence of stages is determined and what the results provided in the stages ought to look like.

In those LCM which mainly aim at controlling the sequence, two types are being discussed most intensively: the early 70'ies waterfall model ([Royce70]) and the spiral model ([Boehm88]).

The waterfall model proceeds more or less sequentially from stage to stage, the stages being *system feasibility, software plans* and *requirements, product design*, etc. until *operations* and *maintenance*. Names for the stages may vary, but characteristically feedback only occurs between neighbouring phases. In the above terminology, once *product design* has started, *feasibility* is never put under question again and so forth. Appealing as this may be from the perspective of controlling large projects, practice has revealed that fully detailed documentation

of the result of some stage may be either impossible or lead to "elaborate specifications of poorly understood (details) followed by ... large quantities of unusable code" ([Boehm88], p.63).

The spiral model can be described by four quadrants characterized by

- determining objectives, alternatives, constraints etc.
- evaluate, resolve, and select, according to the above
- develop, verify the selected alternative
- plan next phase

Activity circulates through the quadrants, going from rough to more detailed realization, always taking into account what has been learned from prior circles. Although theoretically project control is less pronounced here due to planning activities re-occurring after every realization phase, it may be more realistic because it provides more adequate foundations for the required activities.

In those LCM, which prescribe forms of documentation, the forms have to cover the whole variety from a customer's requirement and his organizational environment to optimized code. Since the former two can be assumed to be similar(ly imprecise) and to be dealt with a set of methods and by persons, who only marginally touch on EDP detail, we suggest for this analysis to start with the first phase (resp. its methods and final document) which deals with the "contents" of the to be solved application program and to apply the criteria to this and to the subsequent phases.

3.1 Complexity of the First Language

3.1.1 Software Engineering

Notations such as [Ross77]'s SADT™[4] [5] graphs (with [Teich77]'s PSL/PSA support tools[6], [Yourd89]'s DD's, DFD's, ERD's, and STD's[7], [Bell77]'s RSL,[8] and [GRASPIN] describe data types and data and control flows.
Data types stay far below the expressiveness of logical (e.g. Horn clause logic or 1st order predicate logic) theories, dynamic types as in polymorphism, or full-fledged frame languages ([Bobro77]). RSL data descriptions for example cover non-recursive records. The wide spread E-R model lacks negation and any expressions for general properties [Motta89; Serna89].
In those approaches supporting concurrency, specification of dependencies between processes comes in as an additional element, mostly realized by some extension of the Petri Net notation such as Pr/T nets[9] ([Genri81; Braue87; Hilde89; Angel90]). Essentially, these notations correspond to a partial ordering

---

[4] Trademark of SofTech, Inc, Waltham Ma.

[5] Structured Analysis and Design Technique

[6] Problem Statement Language; Problem Statement Analyzer

[7] Data Dictionaries, Data Flow Diagrams, Entity-Relationship-Diagrams, State Transition Diagrams

[8] Requirements Statement Language

[9] Predicate/Transition nets

of the processes, which can still be dealt with algebraically ([Hilde89]), although recursive records start to play a role. Furthermore, a procedural semantics for Pr/T nets has been provided by [Nemet88; Schoe89; Stuck89].

In all cases graphical presentation plays an important role, as for instance in [Dachl87].

3.1.2 Knowledge Engineering

The most advanced KEngg method, whose original version can be understood as truly life cycle oriented, is KADS [10] [11].

The first intermediate product in KADS is the conceptual model. The KADS language for the conceptual model is informal, making use of the full repertoire of procedural control expressions on the task layer (pseudocode notation), identifies individual processes ("knowledge sources") by their names and by the roles of their formal parameters ("metaclasses"), and allows KL-ONE like (i.e. essentially order sorted logic) notation for the data ("domain theory"). Flexible use of the domain theory for different inferences is supported by forming views of the domain theory ("domain models") and by loosely coupling domain and inference. As outlined in Chapter "Executable Specifications" this can be interpreted as being of expressiveness of order sorted logic with procedural attachment.[12]

This demonstrates the high (and presumably higher) expressiveness of the first language of the KADS KEngg method, compared to the languages established in SE. It still remains an open question, what complexity of an area of application can be treated with such a language. Critique for any such attempt comes from two directions. [Partr87] make the *fundamental* claim that AI problems are characterized by being essentially non specifiable, i.e. no notation would help. [Angel90] emphasize the *technical* problem of the two conflicting aims of having to modularize on one hand and facing highly intertwined knowledge items, which resist modularization. For the latter, the decoupling of domain and inference as realized in KADS might be a solution.

3.1.3 Other Remarkable Work

Though more being in the field of SE, the approach of [Balze78], plays a special role in the following sense. Natural language is now taken as first notation, i.e. a language of highest possible expressiveness. As to be expected from understanding human communication, such a notation will notoriously be underspecified [Wett91b]. This is being dealt with by attempting to apply context information in order to enhance the initial specification towards a pseudocode

---

[10] There is a less elaborate attempt by [Twine88] based on NIAM, which will not be followed up here, since is has not been followed up by the author.

[11] KADS has originated from an ESPRIT project at University of Amsterdam, and is documented e.g. in [Breuk87; Schre88; Wieli87; Wieli89]. There is a somewhat diverging approach to KADS by the author of this text, which is documented in [Wett90a; Wett91a] and is discussed in detail in Chapter "Executable Specifications"

[12] There is a deviating research activity claiming that meta-reasoning is required to fully cover the informal language elements of the conceptual model, cf. [Akker91]

like one. This allows very high expressiveness of the first language and puts the load on the following processes to enable subsequent use of the document.

3.2 Perspective of the First Language

3.2.1 Software Engineering

Most strikingly, all analyzed SE languages for analysis already take a perspective towards the to be produced computational artifact. They use terms such as

```
DEFINE_ELEMENT_TYPE: INPUT_INTERFACE
```

(RSL),

```
Parameters: DB=-EXBDB NAME=hourly-employee-processing ...
  TRIGGERED BY: hourly-emp-processing-event;
  TERMINATION-CAUSES: new-employee-processing-event ... .
```

(PSL), or

```
box list
   box:"/Editing" identifiers = i1, c1, m1, o1
   ...
```

[GRASPIN]

3.2.2 Knowledge Engineering

KADS conceptual model formulations read like

```
Classify Plant
    obtain-data(object attributes)
    classify (object attributes)
    IF solution
    THEN
        display (object class)
    ELSE
        select-request(attribute)
        display-request
        Classify Plant
```

Except for semi-technical terms such as display-request, which are required to describe interaction with the user, the notation addresses an understanding of the task in the domain. The recursive structure becomes obvious by the identity of the first and the last line. The example contains *what* has to be done to classify (in this case) plants, not *how* to do so. Objects and their attributes are independently modelled in the domain layer, i.e. the to be requested attributes are naturally given by the structure of the domain.

The different perspectives taken in the initial steps of SE and KEngg may have both contents related and historical reasons (which may both vanish over time). Concerning SE, communication software e.g. deals with properties, statuses etc. of communication devices, i.e. with entities in an EDP and not in an outside physical or mental world. Or standard application software deals with adminis-

trative processes, whose essential parts are manipulations in a discourse created according to technical, administrative, dataflow, etc. requirements. On this level it is essential, whether a data object *employee* has a data item *social security number* and whether there are no multiple social security numbers etc. This all has much to do with more or less formalized or at least standardized processes and information sets and little with observations and operations in an outside world. Hence it may appear natural to start modelling these standardized processes in SE - but to model things in the world in KEngg. One could even turn the knowledge level debate into its opposite claiming that for normal application programs, communication protocols, heterogeneous networks, the inward i.e. implementation directed perspective is knowledge level for the software engineer who does the analysis.

But the limits may vanish. Think e.g. of an application which technically is distributed computing and legally involves telecommunications. Due to the possibility of displacing the different computers on different plots of land, (which, just to make obvious the challenge) are owned by different companies, which are partially owned by one holding and partially by some distinct private shareholders) the legal aspects of the communication channels must be understood (and hence modelled) in non-technical but legal terms.

3.3 Analytic Guidance

3.3.1 Software Engineering

The only school that seems to come close to offering analytic guidance and not just language is the one of corporate data processing models, i.e. identification of sources, needs, channels etc. of information. AD/Cycle [AD89] is an example of this group, which is also supported by tools but only covers the restricted though economically essential domain outlined above.

3.3.2 Knowledge Engineering

The crisis of the rule based expert systems, as documented in [Bucha84] and other fundamental research in AI gave rise to attempts to analyze human conceivable models of human problem solving. [Chand86] was the most prominent early one, which much influenced KADS's interpretation models [Breuk87]. Here we find domain independent pseudocode like notations of prototypical patterns of solving routine problems such as heuristic classification, monitoring etc.. These interpretation models are to serve two purposes: To help the KEngr to classify the expert activity as being in accordance with an interpretation model in the interpretation model library, (or at least to identify the closest match) and subsequently to interpret details from elicitation sessions with "slots" to be filled in the selected interpretation model. One can be skeptical about the coverage of the whole variety of application areas by the presently rather limited number (ca. 25) of interpretation models.[13] Furthermore, systematic and reproducible attribution of elicited items has been done in the KADS group on the basis of education and skill transfer between specifically trained researchers and is only just recently being studied on a more objective basis ([Rothenf]). But nevertheless, this seems

---

[13] This has in some way already been argued above when claiming that only for very few areas of application there exist theories. Taking this into account, the interpretation models can be understood as approximate or partial theories.

to be the most elaborate attempt to supply means to make an systematic assessment of an area of application.

3.4 From First Notation to Code - The First Step

3.4.1 Software Engineering

Most life cycle based software engineering still seems to follow the waterfall model. With first notations already directed towards implementation, the character of the first step is a refinement or concretion, but not a change in perspective. Hence, the widely used procedure of distinguishing

- functional model
- behavioural model

and

- physical model

may be practically sufficient. However, for achieving each of these from its respective former ones, guidance is poor and more or less ends up with recommendations such as reference to a set of design elements resp. design alternatives or with desired properties of the implementation, such as coherence within and loose coupling between modules ([Yourd89]).

In the future such properties may turn out to be enforced by CASE tools. But at present the competence of the designer guarantees (if it does) that the behaviour of the implementation is as has been approximately expressed in the first notation. This underspecification of the process has the drawback of impredictible final system behaviour but offers the chance to gradually refine. I.e. the discovery of the spiral model may have been due to permanently encountered need to improve the first document.

3.4.2 Knowledge Engineering

Early ([Hayes83]) and present ([Schre90]) life cycle based KEngg is waterfall oriented, with even less feedback loops than in software engineering. (Newer documents indicate discovery of the feedback power of the spiral ([Schre90], p.60), which has, however, not yet had practical implications.)

In this case full specification and probably formal notation is essential in KEngg, when the application directedness of the first notation is taken seriously. Since then this first document talks in terms which an EDP professional responsible for system design will normally not understand on the basis of its contents (since it was created in order that the domain expert recognizes the contents from the document). Hence the designer has to further use the document on the basis of its linguistic properties. Furthermore, if the analyst supplies a (formal) full specification, he is automatically in charge of making all contents related decisions (which is appropriate) instead of leaving these decisions open and having them happen more or less erratically during the work of the designer.

The KADS conceptual model aims at being a ***non-formal*** full specification of the application perspective. The risks of the informal character of the conceptual model are attempted to cope with by detailed information about a language (DDL - Design Description Language, [Schre90]) to use for design from conceptual models. It is meant to offer a functional and a behavioural model which puts it much in the spirit of classical software engineering approaches, see above). ER, enhanced by some semantic database modelling is suggested for the KADS *do-*

main layer. KADS *inferences* are functionally described by names of methods realizing the required inference, plus assignments of elements of the data model to metaclass names. The behavioural description essentially adds more detail about the way how to use the selected method (forward/backward etc.). KADS *tasks* are described by mutual dependencies (used inferences, pre-conditions and post-conditions).

The transition from analysis to design documents ends with statements that the attributions between conceptual model elements and design elements are "usually rather straight forward" ([Schre90], p.61), i.e. prescriptions how to proceed are vague and up to the KEngr, as they have been up to the SE above.

4.0 Executable Specifications

An important special case of the first notation is an executable specification. I.e. the first document is written down in a formal language of which a procedural semantics is available. A number of the arguments of the previous chapter still apply, so this is to carve out those which are either different or can be put more sharply with the additional property of an executable specification.

4.1 Complexity and Perspective of the Language

The requirement to execute the specification on one hand sets upper limits to the expressiveness but on the other hand puts specifications on a solid ground. Nevertheless there are differences, which may be due to the inherent complexity of the areas of application of software engineering and knowledge engineering.

4.1.1 Software Engineering

The details given in the chapter on LCM's also hold here. Execution of the first notation is done as one aspect of [GRASPIN] where term rewrite techniques are applied for verification purposes of TLS, i.e. an Pr/T notation with further algebraic properties.

Some high end approaches have a programming in logic background. They take up the early hope put in PROLOG that it might be the longed for formalism where you can write down the "What" without having to specify the "How" of software products (and which has been contaminated by the double nature of PROLOG as decidable subset of first order predicate logic ***and*** universal programming language). Differences concern the question how to deal with incomplete specifications ([From90]), how to use proofs as programs ([Schwi90]), or how to use a chosen general formalism in application projects ([Stadl90]).

On the borderline to Program Generators (see below) is [Cheng84]'s MODEL, which on the one hand specifies in a language comparable to those languages used here, but on the other hand generates PL/1 code.

Executing a specification often comes close to providing system behaviour traces for verification purposes. Hence, some of the approaches from that area could also be discussed here. The interesting point for comparison, however, is that given a formal language which appears adequate for use as executable specification, its characteristics are explored and eventually the languages is extended.

4.1.2 Knowledge Engineering

KEngg does the opposite: [Wett90a] takes what has turned out useful for highly expressive informal specification in KADS (the conceptual model) and introduces a formal language that covers this catalogue of required language constructs. This ends up with order sorted logic plus procedural attachment, i.e. a language which only is semi-decidable or, in other words, expressiveness is in one way pushed to its limits. It will have to be investigated if this risk is compensated for by better understanding of the specification by KEngr and expert and better possibilities for evaluation.

4.2 Analytic Guidance

While executable specifications in SE come along as languages (which are not models or theories, see above) with their respective interpreters, the executable KADS conceptual model ([Wett90a]) can draw upon the analytic guidance suggested by the KADS interpretation models (see above). Hence the interpretation models are also being made executable ([Wett91a]). I.e. the mental tool for analysis then also becomes a technical tool for studying procedural effects of data interpretations.

4.3 From First Notation to Code - The First Step

4.3.1 Software Engineering:

In software engineering the most approaches are seen on their own, i.e. for engineering ([Fronh90; Stadl90]) or for verification ([Stadl90]) purposes. Only [Schwi90]) primarily suggests to use behavioural traces of the specification ("proofs") as guidance and selection criteria for program design. In other words: most approaches think of redesign instead of transformation.

4.3.2 Knowledge Engineering:

Very definitely the order sorted logic specification of the KADS conceptual models cannot be used in any way as approximation to the final code. Because for the first the notation is clearly domain directed i.e. there is no room at all for specifying any system requirements or implementation guidance. For the second, any interpreter for such a notation will have to be a high end general theorem prover with all its flexibility but also with its lack of efficiency for subclasses of simpler problems.

5.0 Program Generators

"Given a narrow enough domain ... one can design a special-purpose problem-oriented language and implement a special-purpose *program generator*, which compiles this language." ([Rich86], p. xiii). This has happened both in SE ([Neigh84; Barst86]) and in KEngg ([Musen89], and differently in [Lins90b]) with rather similar results concerning the criteria. Differences are primarily due to the characteristics of the selected domains, but not to deviating approaches in SE and KEngg. To distinguish Program Generators as one approach of automated program construction from executable specifications: Program generators end up with a well constructed efficient code (which can only be achieved in narrow domains, see below). Executable specifications are intermediate results with two important properties (full specification, tool to study future system behaviour),

but often highly inefficient and dependent on subsequent redesign for efficiency, modularization etc.

5.1 Complexity of the first Language

The investigations from SE and KEngg share the characteristic that their languages are far from general-purpose. The way they tailor their expressiveness directly reflects where their to be modelled domains are complex and where they are simple. [Barst86]'s ϕNIX for example, whose domain is interpretation of oil well logs, heavily draws upon sophisticated mechanical models, expressed in algorithms for the respective differential equations. [Musen89]'s OPAL is based on a sterotypical yet expressive control structure (skeletal plans), whose individually to be generated plan instances and their activities can just be filled in slots of the overall plan generator program resulting in cancer or hypertension treatment protocols.

While [Musen89] starts from the inference control related epistemic (for the use of the term epistemic cf. [Wett90b]) concept of skeletal plans as the narrow domain, [Lins90b]'s HYPERKSE is restricted to a certain occurrence of knowledge (for the use of the term occurrence cf. [Wett90b]), namely those of distinctions, according to Kelly's ([Kelly55]) psychology of personal constructs. Arbitrary domains can be dealt with to generate executable BABYLON ([DiPri87]) knowledge bases, but the source of the data have to adhere to an absolutely uniform and simple format. In other words: in all outlined approaches the language is simple under most aspects. Where it is complex, the complexity is well understood in application to the small set of appropriate problem classes.

5.2 Perspective of the first Language

The dedicated nature of the languages entails that they can be both, presented to their users in terms of their applications (i.e. they are domain directed) and allow direct creation of artifacts (e.g. they are implementation directed). This can be seen as the theory of the domains having been used to carve out in full detail the procedural semantics of all relevant domain related expressions, hence allowing their immediate transformation in programming language constructs realizing the semantics.

5.3 Analytic Guidance

In can be argued similarly that, assuming a theory of the domain to be given, the generators present domain models to their users, which should be close to perfect guidance, since the users should have similar models in their minds. This also tends to be similar in SE and KEngg approaches.

5.4 From first Notation to the Code

Obviously this is carried out automatically.

6.0 Discussion

The table on the next page summarizes the result of comparison.

Both similarities and differences are obvious from the table. So we just refine our hypothesis why some specific differences can be observed. KEngg by its genuine nature deals with complex and poorly specified entities in the world. To assess

these, it needs highly expressive language and guidance in analysis. It has to be aware that the processes following the first assessment need a change of perspective, not only of concretion. This is not to say that elements of this don't occur in SE. An argument has been given that with complex integrated areas of application, KEngg and SE have to come together. Nevertheless the state of the art at present seems to be that some aspects are approached consciously and methodologically in KEngg, which just happen in SE.

Of course there are also aspects where KEngg cannot compete at present (and maybe principally): Programming in the small as a method to start from well understood primitive algorithms and to gradually transform them into software products that realize and intended system behaviour, has different flavours in SE and KEngg. The more the first notation is already implementation directed, the more can it be suitable for mechanic transformation. This might be most pronounced in communication and process synchronization software, together with PetriNet descendent notations ([Angel90; GRASPIN]). On the contrary, domain directed documents as aimed at in KEngg, almost definitely will require redesign. It may even turn out that in case of a formal first document comparison with the redesigned second document reveals non-feasiblity of some desired functionality. This can, however, be seen as an advantage because it clearly identifies causes of reduced functionality. Some more detail has been given in the chapter about executable specifications. It may also turn out that structures described in the first document are not replicated in the second, because the change of perspective also suggests to arrive at the desired functionality on the basis of a different structure. Certain central aspects could not be treated in the length of the text, such as evaluation activities (validation, verification), modularization (which might be harder in KEngg than in SE because of the high interrelatedness of expertise), tools (which may be harder in KEngg than in SE, because they would have to cover the transition from informal to formal) etc.. Also, database design, conceptual database modelling, semantic relations in database theory etc. suggest themselves for another comparative evaluation.

Table 1. Comparison of approaches

	LCM		Ex Spec		Prototype		Generator	
	SE	**KEngg**	**SE**	**KEngg**	**see final**	**paper**	**SE**	**KEngg**
complexity of language	*low to medium*	*high*	*medium*	*high*			*as required by application*	*as required by application*
perspective of 1st activity; distance to code	*implementation (abstract properties)*	*area of application; abstract problem solving methods*	*implementation (detailed, partly deficient description*	*area of application*			*domain and application perspective coincide*	*domain and application perspective coincide*
analytic guidance	*marginal*	*detailed, based on model of problem solving*	*marginal*	*detailed, based on model of problem solving*			*by theory underlying the generator*	*by theory underlying the generator*
spec -> code 1st step	*only general recommandations and design elements*	*only general recommandations and design elements*	*different strategies*	*redesign*			*automatic*	*automatic*

7.0 Acknowledgements

M.M. Richter pointed out to me the usefulness of comparing KEngg and SE. W. Schönfeld gave valuable advice about different directions that would be worth studying. A number of people provided me highly actual and well selected study material, among them U. Furbach and Walter Sommer. R. Studer and his coworkers, M. Broy, and several people from the KADS group at UvA served as critical discussant. I'm indebted to all of them that this text could be written. Last but not least I want to thank W. Schmidt for permanent technical support in preparing the manuscript.

References

[AD89] AD/Cycle *Systems Application Architekture: AD/Cycle Concepts* AD/Cycle Publication No. GC26-4531, 1989

[Akker91] Akkermans, H., van Harmelen, F., Schreiber, G., Wielinga, B. *A Formalization of Knowledge-Level Models for Knowledge Acquisition* accepted for Int. J. Intelligent Systems

[Anjew90] Anjewierden, A., Wielemaker, J., Toussaint, C. *Shelley - Computer Aided Knowledge Engineering* in: Current Trends in Knowledge Acquisition, Wielinga, B. et al. (eds), IOS Press 1990

[Angel90] Angele, J., Fensel, D., Studer, R. *Applying Software Engineering Methods and Techniques to Knowledge Engineering* Univ. of Karlsruhe, Research Report 204, September 1990

[Balze78] Balzer, R., Goldman, N., Wile, D. *Informality in Program Specifications* in: Readings in Artificial Intelligence and Software Engineering, Rich, C., Waters, R. (eds), Morgan Kaufmann Publishers, Inc., 1986

[Barst86] Barstow, D. *A Perspective on Automatic Programming* International Joint Conference on AI, copie of the conference are available from Morgan Kaufmann Publishers, Inc., 95 First St., Los Altos, CA, 94055 USA, reprinted in: AI and Software Engineering, edited by C. Rich and R. Waters, Massachusetts Institute of Technology, 1986

[Bell77] Bell, T., Bixler, D., Dyer, M. *An Extendable Approach to Computer-Aided Software Requirements Engineering* in: IEEE Transactions On Software Engineering, Vol. SE-3, No. 1, January 1977

[Bobro77] Bobrow, D., Winograd, T. *An Overview of KRL, a Knowledge Representation Language* Cognitive Science, vol.1, no.1, January 1977

[Boehm88] Boehm, B. *A Spiral Model of Software Development and Enhancement* IEEE Computer, May 1988, p.61-72

[Braue87] Brauer, W., Reisig, W., Rozenberg, G. *Petri nets: applications and relationships to other models of concurrency* in: APN 86, LNCS vol. 254-255, Springer Verlag 1987

[Breuk87] Breuker, J., Wielinga, B., van Someren, M., de Hoog, R., Schreiber, G., de Greef, P., Bredeweg, B., Wielemaker, J., Billault, J.-P., Davoodi, M., Hayward, S. *Model-Driven Knowledge Acquisition: Interpretation*

Models ESPRIT Project 1098, Deliverable task A1, Univ. Amsterdam, Dept. Social Science Informatics, 1987

[Bucha84] Buchanan, B., Shortliffe, E. *Rule Based Expert Systems* Addison Wesley Publishing Company, 1984

[Chand86] Chandrasekaran, B. *Generic Tasks in Knowledge Based-Reasonong: High Level Building Blocks for Expert System Design* IEEE Expert (Fall 1986), 32-30

[Cheng84] Cheng, T., Lock, E., Prywes, N. *Use of Very High Level Languages and Program Generation by Management Professionals* IEEE Transactions on Software Engineering, Vol. SE-10, No.5, September 1984

[Daehl87] Dähler, J., Gerber,P., Gisiger, H.-P., Kündig, A. *A Graphical Tool for the Design and Prototyping of Distributed Systems* ACM SIGSOFT Software Engineering Notes vol. 12 no. 3, July 1987

[DiPri87] di Primio, F., Wittur, K. *BABYLON, a Meta-Interpretation Model for Handling Mixed Knowledge Representation* in: Proceedings of the 7th International Workshop on Expert Systems and Their Applications, Avignon, 1987, pp. 821-833

[Fense91] Fensel, D., Angele, I., Landes, D. *Knowledge Acquisdition and Representation Language (KARL)* accepted to Avignon '91, Expert Systems & their Applications, 1991

[Floyd84] Floyd, C. *A Systematic AI Prototyping* in: Approaches to Prototyping, Budde, R., Kuhlenkamp, K. Mathiassen, L., Züllighoven, H. (eds), Springer Verlag Heidelberg, 1984

[From90] Fromherz, M. *Zur Behandlung von unvollständigen Spezifikationen in Logischen Programmiersprachen* in: N. Fuchs, U. Fuhrbach: Workshop 'Logik als Programmiersprache', GWAI '90, Report 13/90 of Univ. Koblenz-Landau, FRG

[Fronh90] Fronhöfer, B. *PLANLOG Logical and Procedural Programming* in: N. Fuchs, U. Fuhrbach: Workshop 'Logik als Programmiersprache', GWAI '90, Report 13/90 of Univ. Koblenz-Landau, FRG

[Gaine90] Gaines, B., Shaw, M. *Cognitive and Logical Foundations of Knowledge Acquisition* in: 5th AAAI- Knowledge Acquisition for Knowledge Based Systems Workshop, Canada, November 1990

[Genri81] Genrich, H., Lautenbach, K. *System Modelling with High-Level Petri Nets* in: Theoretical Computer science 13 (1981), North Holland

[GRASPIN] *Final Document of GRASPIN* Report of the ESPRIT Project 125

[Hayes83] Hayes-Roth, F., Waterman, D., and Lenat, D.(eds) *Building Expert Systems* Addison Wesley, Reading Ma. 1983

[Heise89] Heisenberg, W. *Ordnung der Wirklichkeit* München, Piper, 1989

[Hilde89] Hildebrand, T., Tréves, N, *The Use of Net-Based Graspin Tools For Monetics Applications* technical report of the GRASPIN project, ESPRIT Project 125

[Itzfe89] Itzfeld, W. *The GRASPIN Project and its CASE Environment Prototypes* Final Review, Gesellschaft für Mathematik und Datenverarbeitung (GMD), Institut für Systemtechnik, Sankt Augustin, September 1989

[Kelly55] Kelly, G. *The Psychology of Personal Constructs* Norton, New York 1955

[Lins90a] Linster, M. *Declarative Problem Solving Procedures as a Basis for Knowledge Acquisition: A First Proposal* in: Technical Report of the GMD, June 1990

[Lins90b] Linster, M., Gaines, B. *Hyper-KSE: A Heterogeneous Integration of Acquisition, Interpretation and Documentation Tools* in: 5th AAAI-Knowledge Acquisition for Knowledge Based Systems Workshop, Canada, November 1990

[Motta89] Motta, E., Simi, M., Attardi, G. *Metalanguage and Databases* in: Foundations of Knowledge Base Management, Schmidt, J. Thanos, C. (eds), Springer Verlag Berlin 1989

[Motta90] Motta, E., Rajan, T., Domingue, J., Eisenstadt, M. *Methodological Foundations of Keats: The Knowledge Engineers' Assistent* in: Current Trends in Knowledge Acquisition, Wielinga, B. et al. (eds), IOS Press 1990

[Musen87] Musen, M., Fagan, L., Combs, D., Shortliffe, E. *Use of a domain model to drive an interactive knowledge-editing tool* International Journal of Man-Machine Studies 26(1987), 105-121

[Musen89] Musen, M., Linster, M. *Automated Generation of Model Based Knowledge Acquisition Tools* Pitman Publishing, London, Research Notes in Artificial Intelligence, 1989

[Nemet88] Nemeth, T., Schönthaler, F., Stucky, W. *Formale Spezifikation und Rapid Prototyping - Flexible Systementwicklung mit INCOME* Report of the Institut für Angewandte Informatik und Formale Beschreibungsverfahren, University of Karlsruhe, no.191, 1988

[Neigh84] Neighbors, J. *The DRACO Approach to Constructing Software from Reusable Components* in: IEEE Transactions in Software Engineering Vol. SE-10 Number 5 September 1984

[Partr87] Partridge, D., Yorick, W. *Does AI have a methodology which is different from software engineering* reprinted with permission from AI Review, vol. 1, no. 2, (1987), pp. 111-20. Copyright Intellect Ltd. in: The foundations of artificial intelligence, Cambridge University Press 1990

[Rich86] Rich, C., Waters, R. *Readings in Artificial Intelligence and Software Engineering* Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1986

[Ross77] Ross, D., Schoman , K. *Structured Analysis for Requirements Definition* in: IEEE Transactions On Software Engineering, Vol. SE-3, No. 1, January 1977

[Rothenf] draft of paper presented at first German Workshop on Knowledge Engieneering and Cognition.

[Royce70] **Royce, W.** *Managing the Development of Large Software Systems: Concepts and Techniques* Proc. Wescon, Aug. 1970, also available in: Proc. ICSE 9, Computer Society Press, 1987

[Schoe89] **Schönthaler, F.** *Rapid Prototyping zur Unterstützung des konzeptuellen Entwurfs von Informationssystemen* Dissertation at the University of Karlsruher, 1989

[Schre88] **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

[Schre90] **Schreiber, G., Wielinga, B., Hesketh, P., Lewis, A.** *A KADS Design Description Language* ESPRIT Project P1098 Univ. of Amsterdam 1990

[Schwi90] **Schwichtenberg, H.** *Beweise als Programme* in: N. Fuchs, U. Fuhrbach: Workshop 'Logik als Programmiersprache', GWAI '90, Report 13/90 of Univ. Koblenz-Landau, FRG

[Serna89] **Sernadas, A., Sernadas, C.** *Abstraction and Inference Mechanisms for Knowledge Representation* in: Foundations of Knowledge Base Management, Schmidt, J. Thanos, C. (eds), Springer Verlag Berlin 1989

[Shadb90] **Shadbolt, N., Wielinga, B.** *Knowledge Based Knowledge Acquisition: The next generation of support tools* in: Current Trends in Knowledge Acquisition, Wielinga, B. et al. (eds), IOS Press 1990

[Stadl90] **Stadler ,R.** *Eine Anwendung der logischen Programmierung: Ausführbare Spezifikation von Directory-Systemen* in: N. Fuchs, U. Fuhrbach: Workshop 'Logik als Programmiersprache', GWAI '90, Report 13/90 of Univ. Koblenz-Landau, FRG

[Stuck89] **Stucky, W., Nemeth, T., Schönthaler, F.** *INCOME - Methoden und Werkzeuge zur betrieblichen Anwendungsentwicklung* in: Interaktive betriebswirtschaftliche Informations- und Steuerungssysteme, Kurbel, K. Mertens, P., Scheer, A. (eds), Walter de Gruyter & Co., 1989

[Teich77] **Teichroew, D. Hershey, E.** *PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems* in: IEEE Transactions On Software Engineering, Vol. SE-3, No. 1, January 1977

[Twine88] **Twine, S.** *Representing Facts in KEE's Frame Language* in: Proceeding IFIP WG2.6/WG8.1 Conference on the Role of Artificial Intelligence in Databases and Expert systems, Guangzhou, China, July 1988

[Twine89] **Twine, S.** *A Model for the Knowledge Analysis Process* in: EKAW89, Third Eoropean Workshop on Knowledge Acquisition for Knowledge Based Systems, Paris July 89

[Wett90a] **Wetter, Th.** *First order logic foundation of the KADS conceptual model* in: Wielinga, B., Boose, J., Gaines, B., Schreiber, G., and Van Someren, M. (eds.), Current Trends in Knowledge Aquisition, Amsterdam 1990 (IOS Press)

[Wett90b] **Wetter, Th. and Woodward, B.** *Towards a Theoretical Framework for Knowledge Aquistion* in: Proc. AAAI-Workshop Knowledge Aquisition, Banff, Nov. 1990

[Wett91a] **Wetter, Th. and Schmidt, W.** *Formalization of the KADS Interpretation Models* in: Steels, L., Smith, B., (eds); Proc. 8th SSAISB Conference on Artificia Intelligence (AISB'91), Leeds, April 1991

[Wett91b] **Wetter, Th. and Nüse, R.** *Use of Natural Language for Knowledge Acquisition - Strategies to Cope with Semantic and Pragmatic Variation* submitted for publication in IBM Journal of Research

[Wieli87] **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

[Wieli89] **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

[Yourd89] **Yourdon, E.** *Modern Structured Analysis* Prentice Hall Inc.,Englewood Cliffs, NJ, 1989

Fifth European Knowledge Acquisition for Knowledge-Based Systems Workshop

EKAW91

Knowledge Representation

Keywords: knowledge acquisition, knowledge representation, knowledge visualisation, frame representation system.

A Knowledge Representation Framework Supporting Knowledge Modeling

Geir Aakvik, Agnar Aamodt, Inge Nordbø

ELAB-RUNIT, SINTEF
N-7034 Trondheim, Norway
E-mail: geir.aakvik@delab.sintef.no
E-mail: agnar.aamodt@delab.sintef.no
E-mail: inge.nordbo@delab.sintef.no

Abstract

This paper points out the advantages of using a flexible, expressive and extendible frame representation system in domain knowledge modeling, and how this system also can be used to design and implement a runnable system. In this way, the 'knowledge transformation gap' between an intermediate knowledge model and an operational one is shrunk. Various types of knowledge are extracted during the knowledge elicitation activity and then structured into an intermediate knowledge model, which forms the basis for system design and implementation. The proposed common representation framework is a frame-based system, emphasizing a thorough explicit modelling of various types of knowledge. A short distance between the internal representation and a user readable format facilitates the use of graphical editors both for knowledge modelling and editing of the target system knowledge. Further, this also facilitates knowledge maintenance methods incorporating semi-automatic learning from experience.

1 Introduction

Knowledge modeling is essential for success in the development of expert systems and several approaches has been suggested (e.g. [Breuker,89], [Keravnou,86], [Steels-90]). In the work presented here we focus on modeling as the process of building an intermediate representation between the knowledge elicited from the experts and the knowledge represented in an operational knowledge base. The purpose of the intermediate model is to capture and structure various types of knowledge from the perspective of the domain and the task of the application system. It also serves as a communication medium between experts and knowledge engineers, as an aid in structuring and describing the domain independently of the implementation and as an aid in making decisions about the target system representation, reasoning strategies etc.

An important problem is then how to transform a knowledge model into computational representation format to be used in a running system. The knowledge modelling activity will typically produce a structure containing different types of knowledge, e.g. concept definitions, causal relationships, functional dependencies, process models, associations, heuristics, previously experienced cases, etc. Ideally, a knowledge representation system, within which this knowledge model is to be represented, should allow for the representation of all types of

knowledge that a domain expert wants to express. There are still several problems to sort out in the knowledge representation field before such a system will be available for the general case. It is important, however, that a representational approach is chosen which minimises the weaknesses of current methods. That is, the representation system chosen for a complex problem should be able to express several types of knowledge, and the underlying inference methods needed to interpret the intended semantical content of this knowledge. This also implies that several types of reasoning, such as rule-based reasoning, reasoning within deeper models, and reasoning from past cases, should be enabled by the representational system. The increased complexity of knowledge structures and reasoning methods needed for future, more competent systems, will increase the importance of high-level tools which are able to bridge the gap between an intermediate knowledge model and an executable one (as emphasized in [Vanwelkenhuysen,90]).

Within the knowledge engineering community, there is also an increasing awareness of the knowledge maintenance problem, i.e. how to enable a system's knowledge to be continually modified and extended over time. Systems that are to operate within a real world environment are difficult to maintain by pure manual methods, and some way to enable systems to learn from experience seems necessary. Such methods will typically be semi-automatic, combining machine learning techniques and manual methods. Sustained, continous maintenance and improvement of knowledge-based systems will therefore benefit from tools that make the knowledge representation transparent to the user. A system that addresses this problem is described in [Aamodt,90].

The content of the paper is as follows: The paper first presents a model of the knowledge acquisition process, and points out the role of an expressive and extendible representation system in developing an intermediate knowledge model and in moving from such a model to a target system representation. Then the structure and main components of the representation system SFR (Sintef Frame Representation) are outlined. This is followed by describing a knowledge modeling tool based on the representation system, emphasizing visualisation of knowledge structures and ease of modification. Finally, an example of a system developed and implemented within this modeling and representation framework is discussed.

2 Knowledge Modeling and Systems Development

The METAKREK project ([Sølvberg,88], [Nordbø, 89], [Aakvik,88]) addresses the problem of knowledge acquisition, and developed a methodology for knowledge base modeling and design (METAMETH) and a tool-kit for knowledge base prototyping and testing (METATOOL). The methodology makes the knowledge acquisition process more efficient, puts structure upon the process and provides a set of optional methods and techniques. The importance of knowledge modeling is emphasized and modeling at different levels is proposed. Development of a graphical interface for visualisation of knowledge is a major achievement of the tool-kit.

In figure 1 the different stages in the METAMETH Life Cycle Model (LCM) are shown. The stages are based on different proposals to life cycle models (e.g. [Hayes-Roth, 1983], [Grover, 1983] and [Hayward, 1987]), and are described in [Sølvberg, 88].

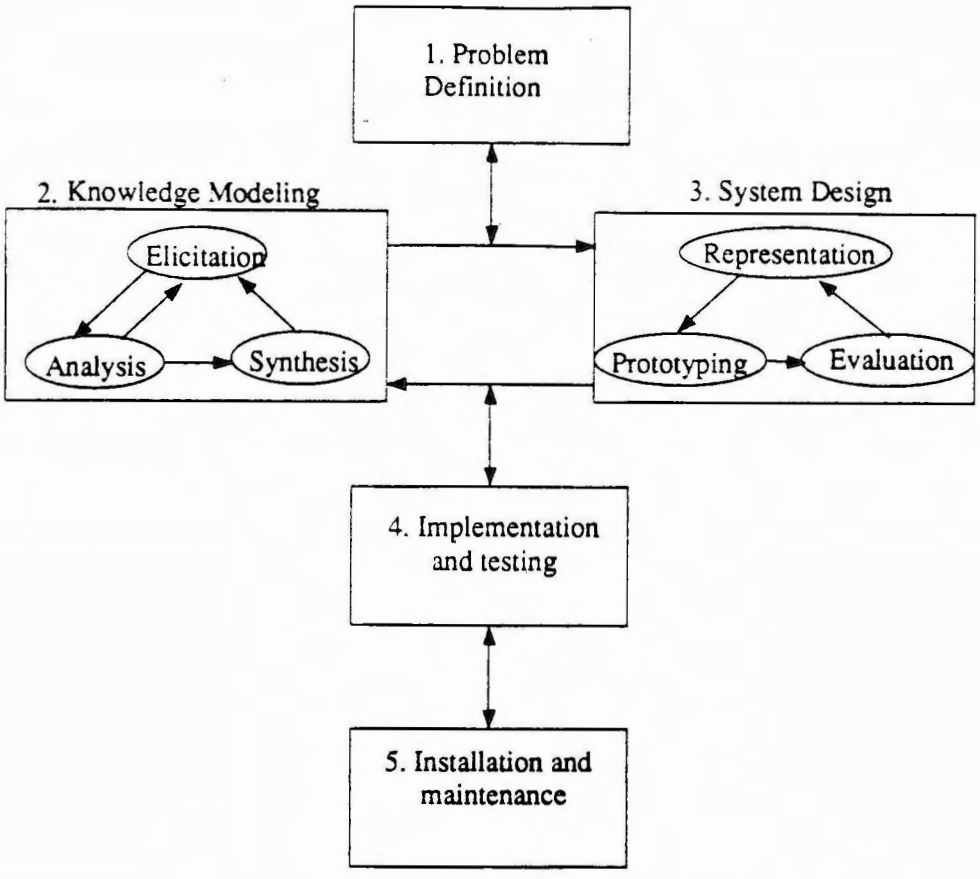


Figure 1 The METAMETH Life Cycle Model

Knowledge Modelling

The goal of the Knowledge Modeling stage is to elicit knowledge and develop models of the domain independently of implementation decisions. The output is a set of models on which the system design is to be based. The term *knowledge model* may be viewed as an integrated network of concepts and relation types — where concepts includes partial models, heuristic rules as well cases, and object level as well as control level knowledge.

The knowledge elicitation process attempts to 'extract' knowledge from a human expert, i.e. to identify concepts, relations, heuristics, dependencies, etc, relevant for the application domain. This initially unstructured knowledge is analysed and structured into what is referred to as an intermediate knowledge model. The purpose of this model is to capture and structure various types of knowledge from the perspective of the domain and the task of the application system, without being constrained by implementational limitations. The knowledge modeling activity is an iteration between elicitation, analysis, and synthesis, where the latter includes the actual building of the intermediate model. The result of this intermediate stage should be a model of domain knowledge and problem solving methods at the knowledge level, i.e. at the level of knowledge content - as opposed to the symbol level, which deals with the computational manipulation of symbols that represent knowledge.

The feedback from analysis and synthesis to elicitation highlights an important property of the knowledge acquisition process: A person's mental model is not a fixed or complete body of knowledge ready to be 'transferred' into some explicit representation. Therefore,

knowledge acquisition should be viewed as an interactive modelling process, where the expert's knowledge - or at least the explicit expression of it - is gradually expanded and refined during the acquisition process. This is sometimes referred to as the cooperative view, or constructive view, to knowledge acquisition [Morik,89]. It follows that knowledge acquisition methods and tools should provide a flexible environment for updating, enhancing and revising the model currently being developed, and not be constrained by, for example, a strict top down refinement strategy. A knowledge acquisition environment - including manual as well as (partially) automated methods - should be an assistant that enables a user to continually restructure and further develop his/her mental model of the domain. Thus, an important role of a knowledge acquisition tool should be to derive consequences of new knowledge entered, suggest modifications, etc. in order to integrate the new knowledge into the existing structure. During the knowledge integration process, contradictions may be identified within various types and levels of knowledge. Some contradictions may be resolvable by the system itself, some may need to be resolved by the user. This interaction is an example of a type of feedback that may improve the person's mental model of the domain, which in turn leads to an improved knowledge model within the system.

System Design

The difficult and challenging task of the System Design and implementation phase is to transform the intermediate model into a computationally tractable model, i.e. into a representation and an accompanying program that is able to solve the problems that the system is aimed at. In addition to problem solving, the computer model should be able to take advantage of its experience by continually extending and refining its knowledge content as new problems are solved. Design and implementation involves choosing or developing languages and tools for knowledge representation, and for programming the reasoning and learning components.

The goal of this task is to work out a system architecture and design principles for a successful implementation. This results in a number of decisions being taken regarding representation, reasoning, uncertainty, man machine interface etc. and in small prototype implementations of sub-models being produced for testing.

A knowledge model should be viewed as a single, tightly coupled network of concept definitions, heuristic rules, past cases, procedures, methods, and strategies. The nodes in the network are concepts and links are relations. Relations are also defined as explicit concepts (e.g. what it means to have colour). Cases and rules are integrated into the network by having their descriptors explicitly defined in the conceptual network.

3 SFR — A Frame Representation System

The representation formalism SFR (Sintef Frame Representation) forms the basis for an **expressive, flexible and extendible** representation system, aimed at facilitating an explicit and thorough modelling of relevant knowledge types. It is based on the knowledge representation in METATOOL [Aakvik–88, Sølvberg–88] developed in the METAKREK project[1] and CreekL [Aamodt-91], designed for enabling representation of multiple knowledge types, such as deep models, rules and previously solved cases. The SFR is also the basis for a frame representation system within the Acknowledge project (Esprit project 2576). SFR is

[1] METAKREK - Methods for Acquisition and Representation of Knowledge is funded by NTNF - IT.0225.16935. The project ran from 1986 to 1988 with a total effort of 5.5 man-years. In addition a number of students have participated in the research by doing project work and master theses. A Ph.D. in machine learning was conducted within the project

a frame-based system, ie. all types of knowledge are represented in frames ([Minsky,75], [Bobrow,77], [Greiner,80]).

Representational framework

A knowledge model represented in SFR is viewed as a semantic network [Aamodt,90]. The nodes and links are explicitly defined in their own frames. Each node corresponds to a concept in the knowledge model, and each link corresponds to a relation between concepts. A frame represents a node in the network, ie a concept in the knowledge model. Each concept is defined by its relations to other concepts, represented by slots in a concept frame.

A concept is either en entity or relation. A relation is the link in the relationship of two or more concepts. (An attribute relation is commonly regarded as a unary relation, describing an intrinsic property of a concept.)

- Concepts are represented as a network of frames
- Concepts are either entities or relations
- Entities are physical or abstract objects of a domain
- Relations are named associations (links) between concepts

The inference methods operating on the semantic network of frames are typical frame language mewthods like property inheritance, frame matching (concept recognition) and constraint propagation.

A set of basic frame operation functions are defined. They all operate on both relation- and entity-frames.

A **frame** is a three-level structure of *slots*, *facets* and *value-expressions*. This structure contains a framename and set of slots. A slot is a structure containing a slotname and a set of facets. A facet is a structure containing a facetname and a set of value expressions. A value expression is a structure containing the value itself (proper value), a justification for the value, a timestamp and an optional parameter:

```
frame := framename {slot}
slot  := slotname {facet}
facet := facetname {value-expression}
```

A **slot** represents both *attributes* and *relations*. Attributes describe the frame holding any type of value, while relations relate the frame to other frames in the knowledge base.

A **relation** is a named association between two frames, and is represented as a frame of type *relation-frame*. The relations are described with a set of predefined slots like: *has-inverse*, *used-to-describe*, *from-class*, *to-class*. These slots may also have facets. A set of structural relations are predefined, for example the specialization and instantiation relations: *subclass-of*, *instance-of*, *part-of*, *member-of*. The inverse relations are predefined as well, i.e. *has-subclass* and *has-instance*, *etc.*

Within this framework, representational *explicitness* is facilitated by representing relations as a type of concept, ie. the representation of concepts and relations are unified. This is considered as a requirement for thorough knowledge modelling, and an important characteristic of the representation system.

Facets (roles of values) enable us to express types of slot contents, like value constraints, deamons - i.e. functions to be called when a slot is referred to. A facet, thus, describes the type or role of a relationship between a framename and the contents of a slot. Examples are:

- *value* — the actual value

- *default* — a default value
- *value-dimension* — a concept or string expressing the dimension of a quantitative value (eg. mg-per-l, "seconds")
- *value-constraint* — a procedure that defines a constraint on the slot's values; values of this facet are general lisp expressions
- *value-class* — a list of concept clases that any value has to be a specialisation of, i.e. special type of value-constraint
- *inheritance-type* — controls the inheritance of a slot.value pair (e.g default-inheritance, forced inheritance, no inheritance)
- *if-needed, if-modified, if-added, if-removed, if-modified* — procedural attachments that are executed when certain operations are performed on a slot's value

Basic frame operations

The basic frame operations defined are:

- *creating concepts* —creating entity-frames and relation-frames, slots facets, values
- *changing frame contents* — changing frame-names, slotnames, facet-names
- *removing frames and frame contents* — removing frames, slots, facets, values
- *displaying concepts* — get frame structurte, slots, facets, values
- *constraint propagation* — This mechanism checks the *value-constraint* and *value-class* facets of the slot involved. Constraints are inherited down *has-subclass* and *has-instance* links.
- *retrieval of value* — The basic content retrieving function first search the value under the *value* facet. If not successful the *default* facet is tried, followed by the *if-needed* facet, whose function gets evaluated.
- *value inheritance* — There are three types of get-value functions working along subclass and instance links:

    a. First look for a *value* facet, then for a *default* facet, then move up one level up and repeat the process etc.
    b. First look for a *value* facet on the local frame, then move upwards looking for value facets, then down again to the local frame and repeat the process for the *default* facet
    c. Search all *instance-of* and *subclass-of* relations recursively, and concatenate the inherited values. Only the most specific concept along each sub-branch is to be included. Multiple inheritance is performed, excluding values inherited along one branch that violates constraints inherited along another branch.

- *traversing relational links* — A set of functions are defined to retrieve concept structures, and to check properties of concepts traversing specified relation links.
- *spreading of activation* — A portion of the concept structure may be activated by spreading of activation from a start concept. Activation is spread along a user-specified set of relations. A default set of spreading relations are defined, for example: subclass-of, instance-of, member-of, caused-by and their inverses. Spreading is done exhaustively along these relations.

Implementation

SFR is written in Common Lisp, and operational implementations exist on TI Explorer and SUN SPARCstations. Various applications are provided with slightly different "instances" of SFR. By using a subset of SFR, making small modifications or extensions to it, the representation system is adapted to meet the requirements of the actual problem. In this

way it is possible to optimize the performance in accordance with the required complexity of the representation system.

4 Graphical Modeling Tool

Interactive graphical interfaces to the knowledge has proved to be very useful in the acquisition process ([Sølvberg,88], [Kontio,89]). Visualising the knowledge gives a better view of it, and improves the process of building, modifying and maintaining knowledge structures.

The expert(s) may often be unfamiliar with the knowledge representation format itself. Visualising the knowledge structures in a proper way makes it more understandable, and helps both the expert(s) and the knowledge engineer in getting a better view of the knowledge. Our experience also indicates that a graphical view of the knowledge model facilitates the communication between the knowledge engineer and the expert.

On top of the representation system SFR we have realized manipulation tools for the inspection and editing of the domain knowledge base. A manipulation tool is here a graphical knowledge editor, which may be used for viewing the knowledge base and for adding, updating or changing the knowledge. In the METAKRE project we also implemented a prototyping tool, ie. a knowledge validation tool, that visualises the reasoning by building a graphically displayed proof tree.

The frames are structured in a directed acyclic network. They are represented as nodes, and the links represent the relations between the frames. Properties from one frame are inherited by the child frame. This kind of visualisation facilitates direct editing, ie. editing the knowledge by editing the graphical picture.

Main functions:

1. Functions for *displaying and editing the structure* of the knowledge base.
2. Functions for *displaying and editing contents of concepts.*
3. Functions for *displaying and editing contents of relations.*
4. *Screen handling* functions (reshape, clear, refresh, zoom, etc.)

New nodes and relations are created, changed or moved by editing directly in the graphical picture. While navigating in the graph it is possible to select and edit single concepts and then adding/changing/removing slots facets and values. It is also possible to show a *full frame*, ie. a frame with inherited values included. Figure 2 shows a snapshot of the knowledge manipulation tool developed in the KNOWIT prosject, (further described in Section 5)

The transparency of the frame representation makes it easy to print the content of a knowledgebase in a user readable format. A set of pretty-print functions/macros is defined to enable printing of the whole knowledgebase, single frames, frames along specific relation links, etc. This printing utility has proved to be very useful both the modelling and the prototyping stages.

Tool implementations

Knowledge manipulation tools are provided in the prototypes developed in the METAKREK [Sølvberg,88], KNOWIT [Sølvberg,91] and ACKnowledge [Marty,89] projects. They have the same basic functionality, but certain adaptions is made caused to slightly different requirements. Further they work in different software environments, ie. the implementation platforms are not the same.

However, using CommonLisp as the main programming language and the same basic knowledge representation system SFR, makes it possible to reuse the main parts of the tools.

METABUILD is developed in the METAKREK project. Running versions exist both on a TI Explorer and SUN SPARCstations. The graphics are based on the standard graphic features in the Explorer Lisp environment, and the Window Tool Kit (Sun Common Lisp extension), respectively.

KNOWman (KNOWledge MANipulation tool) is developed as a supplement to the knowledge based information retreival system KNOWIT, for the inspection and editing of the domain knowledge base. The graphic is based on the Window Tool Kit (Sun Common Lisp extension), and it runs on Sun SPARCstations.

FRED (FRame EDitor) is developed in the ACKnowledge project. This tool works with the KEW infrastructure [Anjewierden,91] and takes advantage of an object management system (OMS) and an user interface management system (UIMS) provided by this infrastructure. The UIMS is based on PCE, an object-oriented system for the development of graphical and menu-driven user interfaces for symbolic languages [Anjewierden,90].

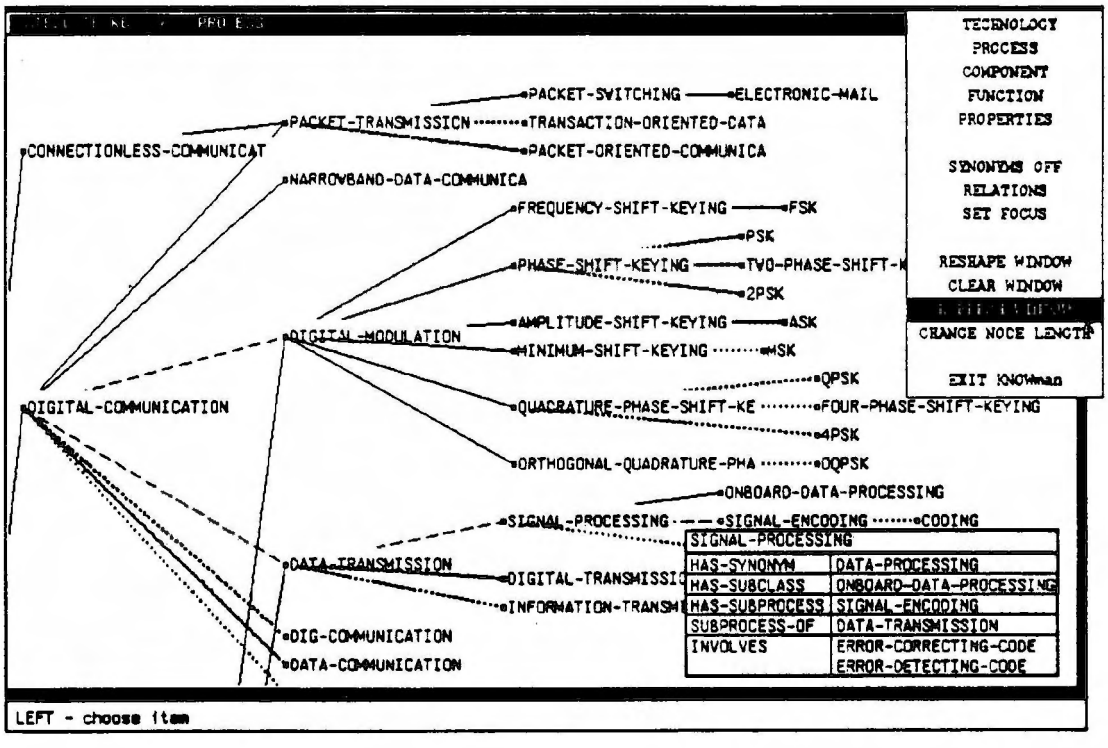


Figure 2 Knowman Editor Window

5 Application — Knowledge Based Information retrieval (KNOWIT)

KNOWIT[2] has been developed as a knowledge-based front-end to the commercial information retrieval system ESA-QUEST, in order to increase the relevancy of the information being retrieved during an ordinary search session. The system is based on a thorough, explicit knowledge model which enables a *focused, semantic matching* process. The knowledge model and corresponding semantic spreading method represent a novel approach to knowledge-based information retrieval, and is particularly aimed at retrieving relevant data sets within very large data collections.

The result of the semantic spreading process is a list of activated nodes (terms). The final result is a complete search query that consists of the input terms from the user and a number of additional terms also considered relevant within the current context.

The subject domain for the implemented prototype is *sattelite communication*. The system makes use of two types of knowledge. The primary knowledge is the subject domain knowledge, but the system also needs some control knowledge for the search and spreading process.

Knowit is implemented in Sun Common Lisp on a SUN workstation. It communicates with ESA-QUEST, which runs on an IBM-compatible main-frame. The subject domain knowledgebase of the prototype contains appr. 1150 domain terms, represented as a semantic network. The knowledgebase is developed with principles from METAMETH Knowledge Engineering Method ([Sølvberg,88], [Nordbø,89], Vestli,89]) using the frame-based knowledge representation language CreekL [Aamodt,90].

The architecture of the information retrieval system with a knowledge-based front-end is shown in figure 3. The user is given a knowledge-based support system for assisting in the retrieval process. The retrieval system itself is unchanged, and the results from the retrieval system are presented to the user as usual.

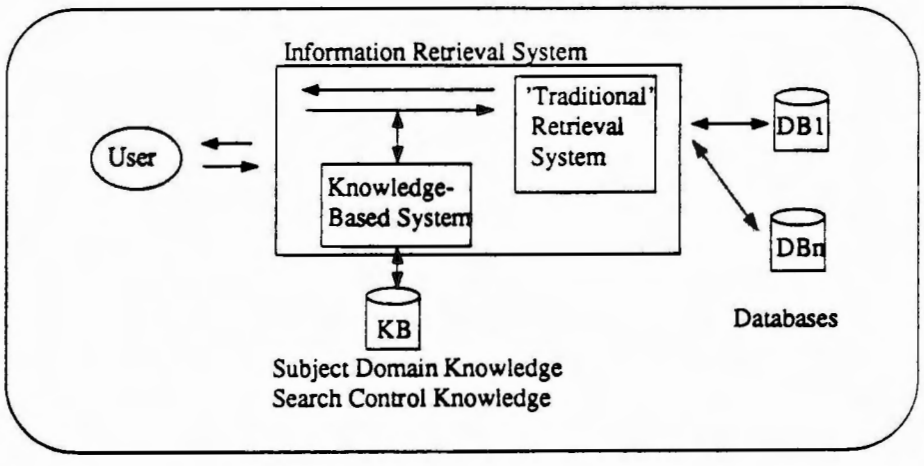


Figure 3 Architecture of an Information Retrieval System with a Knowledge-Based Front-end

Subject Domain Knowledge

An important component in the system is the knowledge model which holds the subject domain model (e.g. *satellite communication*) and the search control knowledge. The subject domain model is a thorough description of the domain, and may be viewed as an active,

[2] This work was done under ESA contract AO/1-2096/HGE-I

extended thesaurus. It is particularly important to note that tis knowledge model contains a richer set of *relations* than a thesaurus, enabling a thorough description of the knowledge *contents*.

The subject domain knowledge is represented as a semantic network, i.e. a network of concepts (e.g. *satellite, wave attenuation, transponder*) and relations (e.g. *has-subclass, involves, has-part*) between them. A node in the network represents a concept, while a link between two nodes represents a relation between two concepts. The (semantic) relations define the meaning of the concepts (terms) in the domain model. That is, the meaning of a concept is defined entirely by the concept's relations to other concepts.

To compose an extended query from the query given by the user, the knowledge base is used to find other terms that seem relevant to the user input term. In this search for other relevant terms, the relations in the knowledge base between domain terms play a key role. This search in the knowledge base for relevant terms to include in the query is referred to as *semantic spreading of activation*[3] within the network of terms and relations (i.e. the knowledge base).

A knowledge-based search is performed by activation within the semantic network representing the subject domain knowledge. When an input term is given, the search module spreads the activation along certain relations. The main tasks are to select the appropriate relations, to select which nodes to expand next and to determine when to stop. These tasks are solved partly with a mechanism based on numeric strengths and partly with control rules.

During the modelling of the subject domain, a set of useful semantic relations were defined. Each of these relations has been given a particular numeric strength. In addition to the numeric strength of the predefined relations, all combinations of two relations are also given a numeric spreading strength. These numeric strengths are used to compute the strength of the activated nodes during the spreading process with a certain formula for combining node strengths from different paths. At each stage in the semantic spreading, the node with the highest strength is expanded further (i.e. a best-first search is conducted).

On top of this numeric machinery, there is a set of rules controlling the search process. Possible actions invoked by control rules are: selecting a subset of relations, limiting the number of levels to spread to, changing control parameters to give more recall or more precision.

The result of the semantic spreading process is a list of activated nodes (terms). Each node in the list has been assigned a certain numeric spreading strength. The terms with a strength above a certain threshold value are included in the set of relevant terms that are returned from the knowledge-based search module and sent to the information retrieval system.

Figure 4 shows a small part of the knowledge base. Note that the dimensions are represented as nodes in the network and that all the relations also have inverses.

---

[3] This technique is a refinement of the method for 'goal-focused spreading of activation' suggested in [Aamodt-90].

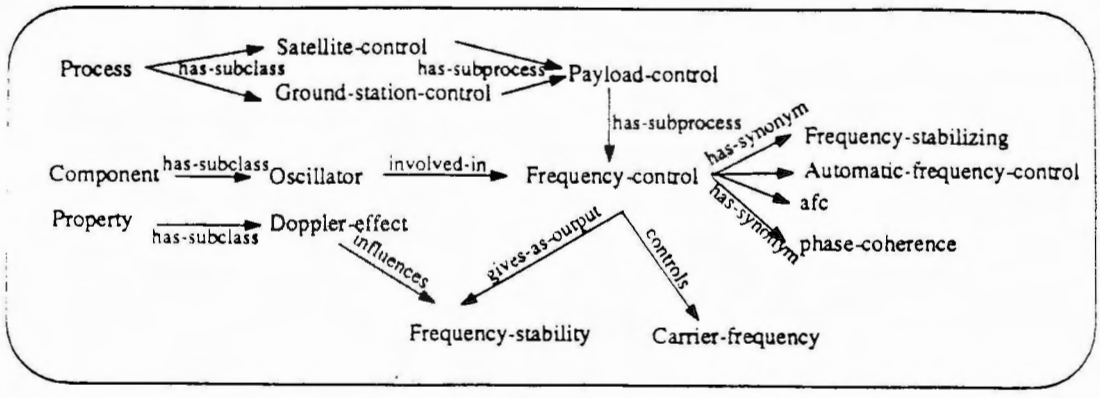


Figure 4 Small Part of the Satellite Communication Knowledge Base

Knowledge modeling in KNOWIT

The development of the knowledge base in this application followed the METAMETH method. The development team consisted of one domain expert (satellite communication) and two knowledge engineers. The main source of information in the initial phases was the existing data base thesaurus (for the NASA data base) which contained a classification of a controlled vocabulary that are used in the indexing of new items in the bibliographic data base NASA. The thesaurus has a few predefined relations between the terms (e.g. generalization, specialization, related to), but those was found to general to be used for knowledge based search.

The first part of the modeling activity was to identify the terms in the thesaurus that was within Satellite communication, to identify the relations in the thesaurus and extend these with more specific relations to express the actual relationships between the terms. An important part of this was then to express the terms and the relations in the representation language in order to test out whether the relations was specific enough to be used for knowledge based search. The spreading activation mechanism and the choice of relations was then done iteratively.

When the set of relations was determined and the main structure of the knowledge base was decided, more of the knowledge in the domain was inserted into the knowledge base and a more extensive testing of the model was done.

Experiences from KNOWIT

The development of KNOWIT has comfirmed that the idea of having a basic representation system intended to be modified to meet the requirements of a specific application is feasible. The focus of the knowledge modeling phase was on formalizing the knowledge according the experts opinion, and tailoring the language instead of tailoring the knowledge to a chosen representation language, which are often the case when using commercial available shells.

Our experience is that SFR was easy to modify in this application. The changes in KnowL (the final representation language in KNOWIT) compared to the modeling language was done in order to better fit to the reasoning methods of KNOWIT, where the algorithm for spreading of activation was modified for this particular application, and a procedure for using control rules was developed.

The representation language was useful in the complete modeling phase, for testing out the structure of the knowledge base to representing the complete knowledge base in the running prototype. It is interesting to observe that the flexibility of the language made

it useful even in the earlier stages of knowledge modeling, before the structure of the knowledge base was fixed. It was possible to test out different kinds of structure in the early stage and experiment with the activation algorithm to find the best way of organizing the knowledge base.

One of the questions we had in mind during the project was whether the efficiency of the language was sufficient when the knowledge base grew, since there is often a trade off between expressive power and computational efficiency. The experience from this application is that the efficiency of the language was sufficient even with a knowledge base of 1150 frames, interconnected by 31 relation types..

6 Conclusion

The frame-based representation system SFR has been advocated as a representational basis both for knowledge acquisition tools and for target system representation. Primary advantages of a frame-based system for knowledge modeling are user-readability and expressiveness. Advantages for implementing operational systems are object-oriented basis of frame systems, enabling a thorough representation of various types of knowledge in a structured way. The focus of this article has been to show that an expressive and extendible frame system is a means to attack the problem of transforming — both ways — between knowledge models developed during knowledge acquisition and a runnable target system.

References

Aakvik,88: Aakvik,G., Vestli,M., Nordbø,I., Sølvberg,I. Amble,T., Aamodt,A.:
METAKREK: METATOOL documentation. ELAB-RUNIT Report STF14 A88055. SINTEF, Trondheim, 1988.
SINTEF Report STF 40 F90201, Trondheim, December 1990

Anjewierden:91: Anjewierden,A. et. al.:KEW Infrastructure Documentation.
ACK-Uva-T3.2–DL-001–A. University of Amsterdam, 1990

Anjewierden:90: Anjewierden,A., Wielemaker,J.: PCE Reference Manual.
University of Amsterdam, 1990

Aamodt,90: Aamodt,A.:
Knowledge-Intensive Case-Based Reasoning and Sustained Learning
Proceedings of the 9th European Conference on Artificial Intelligence
Stockholm, August 6–10, 1990

Aamodt,91: Aamodt,A.:
A Knowledge-Intensive, Integrated Approach to Problem Solving and Sustained Learning. University of Trondheim, Ph.D. dissertation NTH 1991:27.
IDT-rapport 1991:7. Trondheim 1991.
Proceedings of the 9th European Conference on Artificial Intelligence
Stockholm, August 6–10, 1990

Bobrow,77: Bobrow,D, Winograd,T.:
Knowledge Indexing and Retrieval — The
An overview of KRL, a knowledge representation language.
Cognitive Science, Vol. 1, no. 1, 1977. pp 3–46.

Breuker,89: Breuker,J., et. al.:
Model driven knowledge acquisition; interpretation models. Deliverable report Task A1. Esprii project 1098. University of Amsterdam 1987

Greiner,80: Greiner,R., Lenat,D.:
A representation language language.
Proceedings AAAI-80, Morgan Kaufman, 1990. pp. 165–169..

Grover,83: Grover, M.D.: A PRagmatic Knowledge Acquisition Methodology.
Proceedings of IJCAI-83.

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

Hayward,87: Hayward S.A. : How to build knowledge-based systems: techniques, tools and case studies. In Commision of European Communications, ESPRIT'87 Achievement and Impact, Elsevier Science publishers B.V.(1987)

Keravnou,86: Keravnou,E., Johnson,L.: Competent expert system. Kogan Page Ltd 1986.

Kontio,89: Kontio, Jyrki., Lounamaa, Pertti: A graphical framework for knowledge acquisition and representation.
Proceedings of the Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems, EKAW-89. Paris, July, 1989

Marty,89: Marty,J-C., The ACKnowledge consortium: Presentation of the ACKnowledge project. Issued by Cap Sesa Innovation, 13–04–89.
ACK-CSI-WID-OR-001–A. 1989.

Minsky,75: Minsky,M.:
A framework for representing knowledge. In P. Winston (ed): The psychology of computer vision. McGraw-Hill, 1975. pp 211–277.

Morik,89: Morik,K.:Sloppy Modelling. In: K.Morik (ed): Knowledge representation and organization in machine learning. Lecture notes in artifical intelligence, 347. Springer, 1989. pp 107–134.

Nordbø,89: Nordbø,I., Sølvberg,I., Vestli,M.: METAMETH-Methodology for Knowledge Engineering Proceedings of the Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems, EKAW-89. Paris, July, 1989

Steels,90: Steels,L.:Components of expertise. AI-magazine summer 1990, p20–49

Sølvberg,90: Sølvberg,I., Aamodt,A., Bergh,K., Kristensen, O.H., Nordbø,I. Vestli,M.: Study of Knowledge Based Retrieval, Final Report. SINTEF Report STF 40 F90201, Trondheim, December 1990

Sølvberg,88: Sølvberg,I., Nordbø,I., Vestli,M., Aakvik,G., Amble,T., Eggen,J., Aamodt,A.: METAKREK-Methodolgy and Tool-kit for Knowledge Acquisition SINTEF report STF14 A88046, Trondheim, December 1988

Sølvberg,91: Sølvberg,I., et. al.:Study of Knowledge Based Retrieval. SINTEF report STF40 F90201, Trondheim, January 1991

Tong,85: Tong,R.M., Shapiro,D.G: Experimental Investigations of Uncertainty in a Rule-based System for Information Retrieval. Int. J. Man-Machine Studies. 22. 1985.

Vestli,89: Vestli,M., Sølvberg,I., Johnsen,H., Kjøbli,E., Prytz,M., Romslo,I., Nordbø,I., Gleditsch,J.: METAKREK — A Knowledge Acquisition Case Study, Proceedings of 9. Int. Workshop for Expert Systems and their Applications. Avignon. 1989.

Vanwelkenhuysen,90: Vanwelkenhuysen,J., Rademakers,J.: Mapping a Knowledge Level Analysis onto a Computational Framework Proceedings of the Third European Workshop on Knowledge Acquisition

Wielinga,90: Wielinga,B.:Conceptualization of a knowledge engineering workbench. External deliverable ACK-Uva-T1.4–DL-010–A, University of Amsterdam, 1990

Functional and representational integration in knowledge acquisition[1]

Manfred Aben, Maarten W. van Someren and Peter Terpstra

Department of Social Science Informatics
Faculty of Psychology
University of Amsterdam
Roetersstraat 15
1018 WB Amsterdam
e-mail: manfred, maarten, terpstra @swi.psy.uva.nl

Abstract

Knowledge acquisition can be viewed as a process of acquiring knowledge in different representations, using techniques for elicitation, analysis and inductive inference. Informal representations and conceptual models as developed in the project are used as constraints and input for formalisation. A prototype acquisition system is constructed that supports elicitation and inductive inference in various representations. Machine-interpretable knowledge can be interfaced into a distributed representation, keeping the knowledge in the form in which it was acquired and allowing evaluation on test cases. Acquisition tools can be applied at any time and not only in their original form, performing their original task, but also subfunctions can be called and interleaved with other acquisition tools, allowing fine-grained integration.

1 Introduction

Currently the acquisition of knowledge and the construction of a Knowledge-based system (KBS) that is based on this knowledge is performed largely by hand. Acquiring the knowledge for a KBS is a notoriously difficult problem. Various methods and support systems are employed by a knowledge engineer. We can distinguish for example:

Elicitation techniques This refers to techniques for obtaining knowledge from an expert. We can distinguish *structured* and *unstructured* elicitation techniques. The structured techniques rely on a structured representation of the knowledge (e.g. concept hierarchies, repertory grids) and unstructured techniques are more open and usually involve verbal knowledge or knowledge in the form in which it is used by the expert (diagrams, computer output).

[1]The research reported here is carried out in the course of the ACKnowledge project partially funded by the ESPRIT programme of the Commission of the European Communities as project number 2576. The partners in this project are GEC-Marconi, University of Nottingham (UK), Cap Sesa Innovation (F), Sintef, Computas Expert Systems, Veritas Research (N), Telefonica (ES) and the University of Amsterdam (NL). The cases and the knowledge bases on acute cardiac diseases used in this paper were collected and constructed by Li Ling Zheng and Wilfried Post.

Knowledge representation or programming environments Elicitation techniques usually do not obtain knowledge in a form in which it can be used by a computer system. However, in some cases special knowledge representation systems have been constructed that can be used by domain experts, even though they have no experience with computer programming (e.g. [McD90, Mar88, RBW88]). These systems provide some support such as structured editors and make it possible to test the knowledge on test problems, following a edit-debug cycle. Such systems may be very general (such as expert system shells) or specific for some task or even for a task in a domain (e.g. [Mus89]). They may be based on techniques for problem solving and knowledge representation from Artificial Intelligence or they may be specifically constructed for a task or domain.

Knowledge analysis techniques This refers to techniques for analysis of knowledge that is represented in informal languages (e.g verbal data, dictionaries). Analysis involves checking consistency and completeness of terminology, relating pieces of (verbal) knowledge to create a coherent whole, e.g. using predefined models of problem solving processes as in [BW89]. An example of a system that supports this type of analysis is the Shelley system [AWT90].

Machine induction and refinement systems These systems generate or refine a knowledge base from a collection of training problems for which the correct solution is known.

We are interested in a comprehensive approach to knowledge acquisition to be able to construct a knowledge acquisition workbench that includes knowledge acquisition functions from several different tools and that may include representations from different tools, in sofar as the use of these representations is motivated (e.g. because they allow easy elicitation dialogues or because they facilitate induction). In the next section we describe the intended behaviour of a knowledge acquisition system by means of a scenario. Section 3 presents a model of the knowledge acquisition process and finally section 4 presents a prototype system , in which some ideas of the framework are realised.

2 A Scenario

To illustrate the acquisition processes that we consider here, we present a scenario. Because space restriction, it is not possible to illustrate the process in detail and therefore the scenario is in parts abstract.

2.1 Prelimenaries

The task in this scenario is to make a knowledge base, which can be used to classify patients with acute chest pain, as being urgent or non urgent on the basis of givens, which are gathered by interviewing patients [vSZP90]. The classification will be very rough, because no results of laboratory tests are available.

In the initial stage of , experts are interviewed, and documentation is studied. The acquired information is stored in the form of texts of interviews and possibly think aloud protocols. Interviewing and collecting protocols is guided by informal analysis by the knowledge engineer of the extent to which interviews and protocols cover the domain. A list of important terms is compiled from the texts. The knowledge engineer also designs a functional model of the problem solving process. Let us assume that this has the structure of heuristic classification. This model can be compared with think aloud protocols.

A possible next step is to obtain precise definitions of important domain concepts. This can be done by a laddering tool, which elicits a concept hierarchy, or in a procedural context ("how can you tell if attribute A is true of some case?"). This would result in either concepts or some kind of rules. This process is partially determined by the list of terms obtained from the interviews and protocols. New terms that are elicited are added to the lexicon. Elicitation is also determined by the functional architecture, because this indicates e.g. which information can act as input to a classification rule. A workbench should allow the knowledge engineer to keep references between related pieces of knowledge.

The next stage involves implementing the functional architecture, which involves obtaining more detailed domain knowledge. We assume that we have available a set of cases, with symptoms, decision (urgent or not urgent) and some extra information (possible diagnosis, medical findings). This process is largely controlled by diagnosing the problem solving performance of the knowledge that was acquired sofar. This requires this knowledge to be in oprational form. Cases are solved with the available knowledge and the reasoning path is analysed. Comparison with protocols, functiuonal architecture, correct answers and expert data about intermediate steps are used for diagnosis. For example, we invoke a problem solver on the goal `heart-infarction(patient-1, time-1)` (i.e. "Does patient–1 suffer from an heart–infarction at time–1?"). The problem solver cannot confirm the goal. It might be the case that this patient indeed did not suffer from an heart–infarction. But assuming we know (s)he did, we invoke a debugger to asses what part of the knowledge used could be the cause of the failure to proof the goal. The knowledge base contains rules to derive if someone has an heart–infarction at some time. One of its subgoals refers to an attribute `pain-duration` of the concept `heart-infarction`. This concept–attribute knowledge may reside in the acquired concept–hierarchy (which is represented as relations between frames). Analysis suggests several possible causes for the failure. One possible cause may be that there is no such attribute for that concept, although there is an attribute called `duration-of-pain`. Assuming that

we had not yet established in the lexicon that both attributes are synonyms, and we prefer the first naming, the concept editor is used to change the attribute's name.

One possibility is to look back at verbal data, where this concept is mentioned and use this to find the definition, or even to construct concept definitions for some related concepts.

Another possibility is, that there is hardly any knowledge about the concept *pain* and a laddering session is started to identify and define different types of pain. This is connected to the knowledge used for problem solving and now we might try again to see if the goal can be proven. If no proof is found, debugging continues.

Clearly, the rules themselves may be incorrect as well. If this is indicated by the debugger, we can invoke an induction tool, e.g. , to construct new rules from the cases or modify existing rules, based on the cases and the suggestions made by the debugger. In the latter case, faulty rules and proposed modifications may be used as initial hypotheses. Inductive inference from the set of cases can assist in constructing rules for recognising either the diagnosis and decision for a patient from the set of cases that are available, or at a smaller scale, to recognise intermediate conclusions. Induction will use other domain knowledge and be interactive: the expert will guide the construction of rules and will check their validity. Ideally, induction should be integrated with problem solving, applying induction to repair holes in the knowledge base.

2.2 Discussion

This scenario shows, that knowledge acquisition involves different knowledge representation languages and tightly integrated processes for diagnosis, elicitation and induction. The process initially focuses on collecting the terminology and concepts and the functional architecture. Here informal representations play an important role. The next stage is characterised by filling in and refining the knowledge and is guided by diagnosis of the problem solving behaviour that is generated by the current knowledge.

To allow diagnosis and elicitation of relevant knowledge (as defined by the problem solving behaviour), all KA-functions and representations should be tightly coupled, in the sense that at quite a number of moments, they can invoke each other. Similarly our scenario shows that it is desirable that acquired knowledge can be used for different purposes at different moments during , without doing explicit transformation each time we want to asses the quality of the acquired knowledge.

3 A model of knowledge acquisition

The scenario illustrates several important principles of the knowledge acquisition process. The goal of knowledge acquisition is a knowledge base that meets several types of criteria. Some of these criteria apply only to the knowledge base itself (e.g. it should not contain contradictory knowledge, it should be able to solve a

problem in finite time) and others refer to external information (e.g. it should be able to solve a certain class of problems correctly; it should be able to provide the meaning of certain words; it should be able to solve all problems within a specified time; it should be able to supply certain explanations). The data for the acquisition process come in many forms (cases, verbal data, concept hierarchies, modifications to the knowledge base itself, word meanings, a Conceptual Structure, *etc.*). Part of the acquisition task involves generating requests for new data from the expert (or knowledge engineer). More specifically we shall analyse the following aspects of the acquisition process:

KA-cycle The basic process of knowledge acquisition consists of *diagnosis*, i.e. identifying which knowledge is missing; and finding a way to obtain that knowledge, either by *elicitation* from the expert or some other external source or by *inference*, in particular *inductive inference*. The new knowledge is added to the previous knowledge and the next problem is identified.

Consider for example elicitation by Repertory Grid. The grid elicitation system is able to perform (at least) several simple analyses: it can see if the grid is filled (and what is missing), it can see if two constructs can be separated (and which cannot be separated) and it can see if two attributes are equivalent (or not). This information can be used to prompt the user selectively for useful extensions of the grid.

Note that this process has a *closed loop* character, if diagnosis, inductive inference and elicitation use previously acquired knowledge. We shall call this *closed loop knowledge acquisition*, by analogy to *closed-loop learning* [Emd89].

Different representation languages The KA-cycle is complicated by the fact that elicitation usually requires special representations. Examples of representation techniques that are used succesfully in elicitation are eg. Repertory Grid, Concept hierachies and specific cases. This raises a problem, because this requires either translation of these special representations into some central representation language or some special hybrid language. The KA-cycle will have to be adapted to handle this more complex situation.

Formal and informal representations A further complication is, that experts often use informal representations such as natural language or special representations for the domain. For informal representations only a very restricted part of the KA-cycle can be automated, because especially the diagnosis process needs to know the meaning of the representation.

Architecture of the knowledge base Constructing large, complex knowledge bases requires a functional decomposition, using an architectural model. This model provides a natural decomposition of the KA process. Once the functional architecture is established, the KA-cycle can be applied to each component, possibly using functionally adjacent components as constraints. A convenient

assumption is, that each functional component uses a single representation language. In that case the functional architecture of the knowledge base (once it is established) defines the subtasks of the elicitation process.

Although one could look at the process as constructing a multi-language (including informal languages) structured knowledge base, of which an executable knowledge base is only a part, the latter is the goal of the knowledge acquisition task and therefore, one can view the other pieces of knowledge as specifications of this. Knowledge acquisition can then be seen as a problem solving process, concerned with recognising where a knowledge base is lacking in some sense, in particular as violating a syntactic or semantic constraint on the knowledge (as defined by the representation language) or as violating a specification. A problem with the knowledge can be repaired directly by the user, by (inductive) inference or by using some special tool (for elicitation).

The knowledge that is used for knowledge acquisition consists of the knowledge base itself and the current specifications. E.g. elicitation of concept definitions should ideally take into account the effect of a new concept definition on performance of the knowledge base (as specified by cases), the conceptual structure (does it fit or disrupt the structure), the lexicon (is it new), the think aloud protocols (is the concept definition consistent with the use of the corresponding terms in the protocol), *etc.* The existing knowledge is thus used to elaborate or interpret the new knowledge and to check consistency with the rest of the knowledge.

Note that most of the above techniques perform some form of closed-loop knowledge acquisition. E.g. elicitation of a *repertory grid* or a *concept hierarchy* by laddering uses the (incomplete) structure that was elicited to generate new prompts to the user. For unstructured elicitation this is more difficult to see, due to the informal character of such techniques, but it is clear that interviewing an expert often involves analysis of earlier statements by the expert. Knowledge representation or programming systems support their edit-debug cycle by tracing facilities and type checking. These are applied to each new version of the knowledge base (program), which gives a (weak) closed-loop effect.

The model that we presented here should be seen a a task analysis. Some subtasks can be automated and others will have to be performed by a human knowledge engineer or a domain expert. For example controlling the process and requesting complicated knowledge (involving several different types of knowledge) can be very difficult.

It will also be clear now, that existing tools and systems realise specific subtasks, characterised by one or more representation languages, one or more acquisition methods (elictation or induction) and one or more functional architectures. E.g. Repertory Grid elicitation and Concept Laddering basically use a single representation language, with an associated elicitation method. Elementary inductive learning techniques use two components (cases and rules) and a single acquisition technique (an induction algorithm). Integrating such techniques involves relating or integrating the representations and specifying a version of the KA-cycle that applies

to the integrated system. To realise a scenario such as that presented in section 2, this will also involve finer functional integration than using tools as a whole. Eg. during induction, the system should be able to request data from the user or after eliciting part of a concept hierarchy, the system may attempt eliciting related data in Repertory Grid form.

In the next section we describe a prototype of a system that includes representations and algorithms from several acquisition techniques, that are functionally integrated at a fine level of granularity.

4 Realisation

The prototype knowledge engineering workbench ( ) contains several tools. In fact it contains the Shelley workbench, which contains a number of tools designed to support the knowledge engineer in using the methodology [AWT90]. The tools in Shelley that are most relevant in this context are: A *protocol editor* that allows the knowledge engineer to store transcripts of interviews, case studies and written material, and extract from these sources terminology and concepts. The terminology is maintained by the *lexicon editor*. Concepts and their relations and attributes can be maintained by the *concept editor*. The *inference structure editor* is used to design the conceptual model. Shelley offers extensive hypertext– and browsing facillities, that enables the user to relate and inspect all the knowledge acquired on the basis of links between concepts. E.g. occurrences of *pain* in protocols, interviews, concept hierarchy, inference structure, *etc.* are linked together.

A substantial part of the system [TvS88], is also incorporated in our prototype, consisting of: An interactive tool, a many–sorted horn–clause problem solver, a Horn Clause debugger [AvS90] and an facility. The prototype further contains a derivate, a *repertory grid* tool and a *card sort* tool. This is not yet fully integrated with Shelley and therefore occurrences of concepts are not yet linked to those in the Horn clause part of the system.

4.1 Representational Integration

We already indicated in which way informal and formal (executable) representations are integrated: pieces of knowledge that refer to the same concept are linked together, which supports retrieval of relevant pieces for diagnosis or further elicitation. The formal knowledge representation languages should support diagnosis and elicitation, induction or direct editing and debugging (and preferrably all of these). In addition, the knowledge base that contains all knowledge that is acquired, should also support these processes for the entire knowledge base. For example, it should be possible to diagnose a partial Repertory Grid and to request new information from the (expert) user and it should also be possible to diagnose the status of the knowledge that has been acquired sofar, including the Grid. E.g. a Grid may be

incomplete, but the missing bits may not be needed to achieve correct performance of the knowledge base. There are several ways to achieve this integration.

4.1.1 Single representation

We could choose to transform the outcome of each machine–learning or elicitation tool to the chosen representation each time the tool finishes. Thus, the tool maintains its own representation while it runs, but to be able to accept input and generate useful output, transformation algorithms have to be provided that transform from the internal representation to the external one and vice versa. This method has some pros and cons.

- Pros:
  - It is relatively easy to diagnose the entire knowledge base. Inconsistencies and performance errors can be detected relatively easily.
  - The incorporation of "new" techniques "only" requires transformation algorithms to be written and implemented.
- Cons:
  - In general, transformations are not simply reversible, so calling an acquisition tool for elicitation or induction cannot restore the context of the original acquisition. This is a disadvantage especially for elicitation.
  - This will enforces black-box use of tools, which may lead to acquisition of redundant knowledge.

4.1.2 Hybrid representation

Another approach is to construct problem solvers operating directly on the representations used for acquisition and connect these by a kind of message passing. We then have to define "ad–hoc" transformations or interfaces, which determine how a given problem–solver access the different representations. After acquisition representations can be eliminated by transforming the knowledge into the language for the target knowledge base.

- Pros:
  - Obviously this saves some time during elicitation, allthough some extra time may be needed to specify integration points of the representations.
  - The knowledge is constantly available for problem solving/testing
  - If some incorrectness is detected, the source (i.e. tool) of that error may be invoked to remedy the error; the user is instantly "in the right place, speaking the right language".

- Cons:
  - Inconsistency cannot be detected in general. What does it mean for frames and rules to be mutually inconsistent. Moreover, are we to commit ourselves to consistency at each point. One could take a pragmatic standpoint, stating that knowledge is inconsistent if it actually leads to incorrect problem solving behaviour.
  - The interfaces have to be specified, however this is not an essentially different task than specifying the transformation algorithms in the transformation approach. However, providing the information where and when to interface to other representations might prove not trivial.
  - To assess the consequences of changing some knowledge requires full problem solving.

4.1.3 Chosen Representation Scheme

In addition to the options mentioned above, it is possible to invent new representations and to adjust KA-techniques to representations that were imvented for some other purpose. E.g. we adapted the algorithm to induce Horn clauses instead of boolean combinations of attribute-value pairs, as the original version did. This makes it possible to integrate it with EBG and a debugging system. Basically, we chose the hybrid solution, minimising transformations. Therefore our system now contains sorts, Horn clauses, concept hierarchies, repertory grids and several informal representations (lexicon, texts). Shelley and the repgrid tool and the card sort tool share a central object–oriented frame–like representation. Moreover, shelley offers extensive hypertext linking facilities.

4.2 Functional Integration

Another form of integration required for knowledge acquisition, is integration of techniques, or rather, integration of functionalities implied by the techniques. We describe again two possible ways of achieving this type of integration, along with their pros and cons: the "black–box" versus the "fine grained" approaches.

4.2.1 Black box integration

One way of having tools collaborate is to run them one by one, in a statically or dynamically determined order, each one tool until completion, and having the next tool working on its results. The tools are thus viewed as "black boxes" with respect to other tools: A given tool does not "know" — let alone influence — what happens in another tool, it only knows what the other tool produces and what it consumes.

- Pros:

- Integration can be done by defining transformation algorithms, techniques can be used "as–is".
- The techniques' semantics are left intact.
- Their workings are clear and well–described.

- Cons:
  - Quite rigid; it does not really allow much jumping back and forth between tools, since tools have no access to their intermediate knowledge state.
  - Because there no fine–grained KA–cycle is possible, and therefore no fine–grained control over the acqusition procesesses, these are likely to acquire redundant knowledge.

4.2.2 Fine grained integration

Another way to accomplish functionality integration is by "breaking tools open", i.e. specifying them in such detail, that shared and shareble functionalities present themselves. Many induction techniques, for instance, perform some kind of evaluation, and these evaluation modules may be interchangeable, or can be replaced by a special–purpose evaluator.

- Pros:
  - Allows the expert/knowledge engineer to interact with the techniques, at a fine level of control, cutting off unpromising searches, or suggesting promising ways to go. This ensures a real intelligent cooperation between the workbench and its users(s).
  - Flexible configurations of tools may reveal possibilities otherwise to remain unrevealed. Tools will be much more powerfull.
  - Facilitates efficient "smart coding", i.e. no procedures in different tools which have a common functionalitie, have to be duplicated.

- Cons:
  - Lots of possible combinations. It will be difficult to control the acquisition process.

4.2.3 So what do we choose?

The scenario clearly suggests that fine–grained integration is necessary, so aimes for this. Most of this is achieved by using problem solving behaviour as the basis for diagnosis (instead of e.g. consistency checking). The hybrid representation supports continuing elicitation and refinement. A second method consists of the hypertext links between informal pieces of knowledge (and, in the future, formal knowledge structures).

4.3 The system

Below we list the tools contained in the current system, the functionalities provided by each tool, and the representation each tool uses. After that we describe in detail how the representations and functionalities can be tightly integrated in the proposed architecture.

4.3.1 Included tools

- Protocol Editor

  Functionality: The protocol editor offers facilities to edit and store protocols, written material, and other unstructured text. It also offers analysis and hypertext facilities, such that it can be used to identify concepts and terminology of specific interest to the domain. The hypertext facilities make it possible to relate any part of the text to any object in the system.

  Representation: The representation is plain, unstructured text, and links between fragments of text and between fragments of text and any object in the system. Fragments and text are stored in frame–like objects, which we will refer to as Shelley–objects.

- Lexicon Editor

  Functionality: This tool allows the knowledge engineer to identify domain–specific terminology and specify translations and synonyms of these terms.

  Representation: Shelley-objects with slots for name, synonyms, translations, abbreviations and description.

- Concept Editor

  Functionality: The concept editor can be used to enter concepts and attributes to the concepts. It also provides functionalities to structure the concepts in hierarchies. (Multiple) inheritance of attributes is supported.

  Representation: Shelley objects, with the same slots as lexicon objects, plus slots for all related attributes, and relations between these objects.

- Inference Structure Editor

  Functionality: This is a tool to specify inference structures by selecting and/or creating knowledge sources and meta–classes and relating them. The tool offers a choice from a fixed set of knowledge sources, for which it does consistency checking. Thus, each knowledge source may only have one output, and the number of allowd input metaclasses for some knowledge sources is fixed by deinition. The Shelley system also contains an

Interpretation Model Library tool, from which the knowledge engineer may select an inference structure, and refine that to fit the domain. For an extensive description of inference structures, the reader is referred to [BW89, BWvS+87].

Representation: Shelley objects representing knowledge sources and meta–classes, and relations between them.

- Tool

  Functionality: This tool implements an 15–derivate [MMHL86]. This specific implementation is an "open" one, in the sense that the knowledge engineer or the expert has a lot of influence on the way the tool proceeds. (S)he can select promising hypotheses to be persued first, and unpromising searchpaths to be abandoned. Preference criteria may be set such that will prefer hypotheses containing some attribute value pairs. Generalisation and specialisation rules are not fixed, but can be user–defined.

  Representation: The attribute–value pairs, hypotheses and cases are represented as Horn Clauses (Prolog). The representation can handle not only attribute–value pairs, but also relations between objects or attributes.

- Horn Clause Problem Solver and Debugger

  Functionality: The system contains a horn clause problem solver, capable of dealing with sorted unification. The debugger can be used to complete failing problem solving traces, by finding a smallest possible change to the theories used, which would enable the problem solver to succesfully solve the problem [AvS90].

  Representation: Horn clauses annotated with types and an indication of reliability. Reliability is expressed nominally and can be one of *definition, default, observation or theory* [TvS88].

- Explanation Based Generalisation

  Functionality: The system contains functionalities for explanation based generalisation [MKKC86] and explanation based generalisation of failures [SP88]. The problem solver recognises and prefers rules generated by these algorithms.

  Representation: Horn Clauses.

- Tool

Functionality: This is a frame language with inheritance and a classifier. It can assimilate new frames by classifying them and including them in the concept hierarchy at the appropriate place. Procedural attachments can be defined as calls to the Horn clause problem solver. checks several properties of the frame hierarchy, such as identical concepts, violations of the *is–a* relation. This tool is , a ne [BS85] derivate.

Representation: is implemented in Prolog.

- Repertory Grid Tool

  Functionality: This tool enables the knowledge engineer to identify attributes which diferentiate concepts. Attributes are elicited and concepts are scaled over these attributes. The tool provides means to cluster or discern concepts which appear to be similar.

  Representation: Shelley objects with attributes.

- Card Sort Tool

  Functionality: This tool implements the card sort elicitation technique. The knowledge engineer can create cards represnting concepts in the domain, and sort them over values of an attribute, all elicited from the expert. The obtained sorts can be transformed to attribute/value pairs of concepts.

  Representation: Shelley objects

4.3.2 A Scenario

Let us use the system to construct a knowledge base for the diagnosis of acute cardiac diseases, based on observations on a patient. We start out with as initial data a set of interviews in the form of plain text, and a set of cases represented as sets of prolog facts.

The interviews are read into the protocol editor, and we can analyse the protocols to distinguish important concepts and terminology. This can be done by selecting words in the text and instructing the system to append them to a lexicon, or to the concept editor. In the lexicon we can annotate these terms by providing it with translations, abbreviations and synonyms.

After identifying the concepts from the protocols, we can relate pieces of the protocols, called *fragments* to the concepts through hypertext links. This enables us to keep track of were concepts came from in term of elicitation. Next we will use the concept–editor to elicit attributes of these concepts, again by analysing the protocols. These concepts include diagnoses, symptoms, prescriptions, diseases farmaceuticals etc. Again we may link fragments from the protocols as informal

	PED	CST	RGT	CMT	PS	DEB		CED	KLA
PED	hyper direct	hyper	hyper	hyper	-	-	-	hyper	-
CST	hyper	direct	rel direct	rel	-	-	-	rel direct	-
RGT	hyper	rel direct	direct	rel	-	sim	pref hyp	rel	-
CMT	hyper	rel	rel	direct	-	-	-	rel	-
PS	-	call	call	call	direct	direct	direct	call	call
DEB	-	call	call	call	direct	direct	direct	call	call
	-	-	-	-	direct	direct	direct	-	-
CED	hyper	rel direct	rel direct	rel direct	-	-	-	direct	-
KLA	-	call	call	call	call	call	call	call	direct

Legend:
The vertical column indicates the "from" tool in the interaction and the horizontal row contains the "to" tools.
hyper *Objects used in both tools can be related through hypertext links*
call *The representation language of the from tool has facilities to issue calls to the to tool, as to get answers to specific queries to the from tool.*
direct *The to and from tools may share the same datastructures.*
rel *datastructures in both tools may be related over arbitrary relations.*
sim *The rep grid tool's output can be used as similarity criteria, which can be used by the debugger to propose a small modification to the knowledge base to fix an observed problem.*
pref *The rep grid's output can be used by the tool as preference, or relevance criteria, describing which attributes of a concept should prefer when inducing a class description of the object.*
hyp *The rep grid's output may be used as initial hypotheses by .*

Table 1: *Interactions between tools.*

justification for the attributes. Next we will elicit the relations between the concepts, by defining concept–hierarchies, which may be any relation, such as *is–a* and *consists–of* relations. In this way, we have done some initial knowledge acquisition, determining relevant concepts, terminology, and relations between concepts.

The next step will be the identification and description of the reasoning that is performed in the domain. This is done by identification of basic inference steps and knowledge roles, as defined by the –methodology. Again the protocols, which may also include case–studies and think–aloud protocols, form the source of this knowledge, and similar to the way concepts were extracted, we can extract knowledge sources and meta–classes. In the inference model editor, we will structure the knowledge sources and meta–classes into inference structures. These inference

structures, however are rather abstract. In the domain at hand, the inferencing can be modelled by heuristic classification.

The next step will be the "filling in" of the knowledge sources and meta–classes, by determining the actual knowledge which plays the roles prescribed by the meta–classes, and the actual executable knowledge, which performs the inferencing described by the knowledge sources.

In heuristic classification observations are abstracted to symptoms, which are heuristically matched to diseases, which are specialised to prescriptions of medication. The meta–classes involved are observations, symptoms, diseases, and medication. The cases we started out with constitute the observations, thus by linking the cases to the meta–class observations, we make this relation clear. In the same way, we elicited diseases, symptoms and medication from the protocols into concepts and structured them into is–a hierarchies, and again, we can link the metaclasses to these hierarchies.

The knowledge sources will be "filled in" by horn clauses, which are obtained by elicitation. The clauses which do not use intermediate concepts, i.e. lead from for instance observations to diagnosis in one step, can be acquired by the – tool on the bases of cases. The cases consist of descriptions of observations on a particular patient, in terms of observables and symptoms, the diagnosis (disease) and the prescribed medication. These cases form the basis to induce abstraction rules which abstract symptoms from observables, association rules, which associate diseases with symptoms, and prescription–rules which suggest possible medication for specific diseases. The induction tool included in our system is capable of using preference criteria over attributes, stating which attributes are to be preferred in constructing an hypothesis. These preference criteria may be elicited directly, but in the present scenario, we use the repertory grid tool to elicit these preference criteria. The entailment rules produced by the repertory grid may be used as initial hypotheses for induction.

Next we want to asses the problem solving capacity of the acquired knowledge. The system contains a problem solver, which is capable of using executable knowledge to answer queries. This is achieved by an ex[plicit description of how inferences are to be made in each representation, and what role each representation plays in the problem solving process. If we put a query to the problem solver, such as "Does patient X have a coronary disease Y", then we need observations of this patient X, we need the rules induced by which relate observables to specific diseases, and we need the *is–a* hierarchy over diseases to assure that the disease found is in fact a coronary disease. So we need to be able to reason with the cases, which were represented as Horn clauses, abstraction– and heuristic match rules, which were induced by the –tool, and represented as Horn clauses, and concept–hierarchies, which are represented as Shelley frame–like objects.

If the problem solver fails to answer the query correctly, we can use the debugging–system, part of the prototype. When the problem solver fails to give a correct answer, than either the knowledge used in the derivation, or the way it was used was

incorrect. Let us assume that based on the induced rules, and the observations on patient X, we derive that X suffers from arrhythmia which is not a coronary disease. If the answer to the aforementioned query was affirmative, this means that the system incorrectly concluded that arrhythmia is a coronary disease. This means that the hierarchy incorrectly declared arrhythmia to be a subclass of coronary diseases. Since the system has an explicit description were the knowledge comes from, and how it is used, the system will present the user with the concept editor again, and point out were the hierarchy is wrong. In this way, the knowledge can be refined in exactly the same way and representation it was initially acquired in. The user can use the protocols again, by following the hypertext links, to establish if the protocols indeed state that arrhythmia is a subclass of coronary diseases, which it does not.

5 Discussion

The knowledge acquisition system that we described was constructed following these principles:

- analyse existing techniques for knowledge elicitation and machine learning into their functional components
- consider the purpose of a special representation language that is used by a knowledge acquisition tool (eg. natural representation for domain expert or end-user to specify her knowledge, restriction of search space for rule learning)
- apply the functional components to construct a representation system that includes various components that specify the knowledge, in addition to a hybrid representation language

The main achievement is a system , that supports various types of knowledge acquisition, including a prototyping approach (which does not use the facilities for specifying architecture, conceptual framework, *etc.*), an inductive learning approach (which will not use the specification facilities and probably only a single representation language), a -style approach (a top down approach: first build the specifications and only then the actual knowledge base) and an approach based on fixed architectures (which is achieved by fixing the architecture of the knowledge base). Each approach is supported by tools for learning and elicitation, but only as far as the available information allows this.

The system described here is similar to the *generic task* approach [Cha88, Cha89] in the sense that it is based on the idea to use special representations and problem solvers for special subtasks, but there are several important differences:

- one representation can be used for several functions (eg. the concepts in the concept hierarchy can be used for *classification*, but also for *direct information retrieval*)

- the system includes "knowledge analysis" support in addition to executable components. This allows separate acquisition and analysis of the architecture and conceptual structure of the domain.

- for many representations (and therefore for the associated task), there is support for elicitation and induction in the form of structured editors, elicitation protocols, inductive extrapolation.

The current system is a prototype, which needs minor extensions, but also raises some major problems. Minor extensions that we hope to realise in the future are explicit links between the knowledge base under construction and the Conceptual Model. These will be used to support refinement and induction. Another limitation is, that several connections between components of the hybrid language are not defined yet. Several induction techniques will be added to this system, in particular a clustering technique to support construction of concept hierarchies.

A major research problem that becomes visible (and therefore researchable) in the context of this system is the selection of a tool (acquisition technique *cum* representation, because this decision must take into account both the applicability of the acquisition technique (such as comprehensibility to the expert, expressiveness, number of cases) and the architectural consequences for the actual knowledge base. E.g. to decide how knowledge about some concept is to be acquired, the knowledge engineer must consider the applicability of an elicitation tool for this type of concept and she must also consider the effect of integrating the chosen knowledge representation into the entire system. We believe that this will require a global model of the problem solving architecture of the system from an early stage in the acquisition process. For some preliminary empirical work on this problem see [SvS91].

References

[AvS90] M.W.M.M. Aben and M.W. van Someren. Heuristic refinement of logic programs. In L.C. Aiello, editor, *Proceedings ECAI-90*, pages 7–12. Pitman, 1990.

[AWT90] A. Anjewierden, J. Wielemaker, and C. Toussaint. Shelley — computer aided knowledge engineering. In B. Wielinga, J. Boose, B. Gaines, G. Schreiber, and M. van Someren, editors, *Current trends in knowledge acquisition*, pages 41 – 59, Amsterdam, May 1990. IOS Press.

[BS85] R.J. Brachman and J.G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9:171–216, 1985.

[BW89] 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.

[BWvS+87] 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.

[Cha88] B. Chandrasekaran. Generic tasks as building blocks for knowledge-based systems: The diagnosis and routine design examples. *The Knowledge Engineering Review*, 3(3):183–210, 1988.

[Cha89] B. Chandrasekaran. Task structures, knowledge acquisition and learning. *Machine Learning*, 4:339–345, 1989.

[Emd89] W. Emde. Lernen im geschlossenen kreislauf. In *Proceedings Wissensbasierte Systeme, 3. Internationaler GI-Kongress*, pages 72–84, Muenchen, Germany, 1989.

[Mar88] S. Marcus, editor. *Automatic knowledge acquisition for expert systems.* Kluwer, 1988.

[McD90] J. McDermott. The world would be a better place if non-programmers could program. *Machine Learning*, 4(3):nn–mm, 1990.

[MKKC86] T.M. Mitchell, R. Keller, and S. Kedar-Cabelli. Explanation based generalization: a unifying view. *Machine Learning*, 1:47–80, 1986.

[MMHL86] R.S Michalski, I. Mozetic, J. Hong, and N. Larsson. The aq15 inductive learning system, an overview and experiments. Technical Report ISG 86-20, Dep. of Comp. Science, Un. of Illinois, Illinois, Urbana, 1986.

[Mus89] M.A. Musen. *Automated Generation of Model-Based Knowledge-Acquisition Tools.* Pitman, London, 1989. Research Notes in Artificial Intelligence.

[RBW88] B. Porter R. Bareiss and C. Wier. Protos - an exemplar based learning apprentice. *Int. J. Man-Machine Studies*, 29:549–561, 1988.

[SP88] J.L. De Siqueira and J.F. Puget. Explanation-based generalisation of failures. In *Proceedings Of The European Conference On Artificial Intelligence*, pages 339–344, Munich, Germany, August 1-5 1988.

[SvS91] Luis F. Solorzano and M.W. van Someren. Coherent knowledge acquisition needs a functional architecture. In *Submitted to EKAW-91*, 1991.

[TvS88] 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.

[vSZP90] M. W. van Someren, L. L. Zheng, and W. Post. Cases, models or compiled knowledge? - a comparative analysis and proposed integration. In B. J. Wielinga, J. Boose, B. Gaines, G. Schreiber, and M.W. van Someren, editors, *Current trends in knowledge acquisition*, pages 339–355. IOS Press, 1990.

MODELLING FARADAY'S DISCOVERY OF THE ELECTRIC MOTOR: An investigation of the application of a functional database language

T. R. Addis†, D. Gooding †† and J. J. Townsend†

ABSTRACT

A functional database language FAITH has been developed in order to provide a modelling tool that enables an investigator to create a range of models. These models can be based purely on observation or on formal theory or both. This paper describes one model (EXPLORE) formed by examining one day of Faraday's laboratory notes in which he discovered the electromagnetic rotation motor. The model depends upon representing the 'train of thought and action' of an investigator (Faraday) by a sequence of features. These features represent the foci of attention that govern the outcome of an experiment or construal. This model clearly illustrates the balance and symmetery of thought and action in the discovery process. Through modelling other examples of real experimentation such models can be developed into a rigorous theory of the mechanisms of scientific activity; moreover activity displayed by some of the best scientists in history at the peak of their productivity.

1. Introduction

This paper describes in terms of a functional database language (FAITH) an example of situated learning in a material environment, tracing the process that produced the first electromagnetic rotation motor in September 1821. This was invented by Michael Faraday, who is widely regarded as one of the greatest experimentalists in the history of science.

We are not at this stage intending to do anything more than reinterpret the recorded sequence of events (as given by Faraday's notebooks) in terms of FAITH functions. The starting point for considering the functions involved is the work by Gooding (Gooding, 1990a, 1990b). Gooding created a visual language that illustrates the sequence of events that lead to discovery. Gooding's maps display a sequence of operations, outcomes and decisions for each segment of an experiment. They emphasise the investigator's *ontology* (actions, effects, material objects, concepts, methods, techniques, technologies, etc). By contrast, the model developed here also deals with the *structure* that episodes of situated learning should exhibit. It is hoped that through such reinterpretations models of discovery, providing some insight into the nature of scientific discovery, will emerge from our *empirical* study of this and other examples of experimentation.

Our mode of operation has been to examine the contents of the notebooks, the historical context of the experiments and to reproduce the experiments with the original

† Knowledge Systems Group, School of Engineering and Information Science, University of Reading, Whiteknights, Reading, Berkshire, RG6 2AX

†† Science Studies Centre, School of Social Sciences, University of Bath, Claverton Down, Bath, BA2 7AY

methods, using copies of the apparatus (Gooding, 1989a, 1990b). A small set of extensionally defined functions is then proposed that seems to describe the observed and implied processes. These extensionally defined functions are used to describe one or more functions that can, when run as a computer program (an intensionally defined function - the model), reproduce the sequence of observed or implied events.

The model may be adjusted through the creation of different functions or types until it satisfies the empirical evidence. The choice of model will be that which has the greatest capacity to generalize over different scenarios of discovery and yet remains relatively simple. Such a model can generate alternative approaches to discovery. The existence of such paths (see figure 11) confirms the model. The model therefore promises to become a tool for the historian to study simultaneous discovery. The model may eventually incorporate some of the elements of 'discovery programs' to replace some of the extensionally defined functions (e.g. Langley P, 1982). This new model would then provide empirical support to formal models of cognition (in its broadest sense). In any event, these models will expose some of the essential components of the process of scientific discovery.

2. Evolving The Model

One of the most striking results of the 'Gooding maps' is the close interplay between the activities of discovery (experimentation and observation) and the cognitive processing of discovery (construal and insight). Our preliminary analysis identified three functions that would extensionally represent the particular sequence of events described in Faraday's notebooks dated September 3rd 1821. These functions were EXPERIMENTS, CLARIFY and CONSTRUAL.

Each of these initial functions has been extensionally defined and in order to make clear the 'meaning' of each function, the text (and in some cases the diagrams) have been reproduced for the first experiment of the day (see Addis et al, 1990b for more detail). The functions have been created within a *program development environment* CLARITY (Addis et al 1990a). CLARITY enables a user to generate a FAITH program by drawing named boxes and labelled arrows. The diagrams represent 'circuits' of functions linked by typed 'wires'. These 'circuits' can be recursive and represent higher order functions (i.e. functions that take functions as parameters). Models can be developed piece-wise and this flexibility makes CLARITY an ideal tool to create models of discovery.

Figure 1 is a schema for the extensional definition of the function EXPERIMENTS. An apparatus is usually capable of being used to explore a range of phenomena and consequently a set of results is often expected during a particular session of work. However, the creation and use of the apparatus are governed by the particular expectations of the investigator. These expectations are derived from two distinct sources. The first is the need of the investigator to test the consequences of a particular hypothesis. This hypothesis is usually a simple construal derived from perceiving known phenomena in a new way. It is an attempt by the investigator to confirm his or others' observations within an abstract framework; a framework that exists behind the range of observed behaviour. The first input to the function is a reference to the observed phenomena that imply this framework.

The second source that governs the outcome of a particular range of experiments is

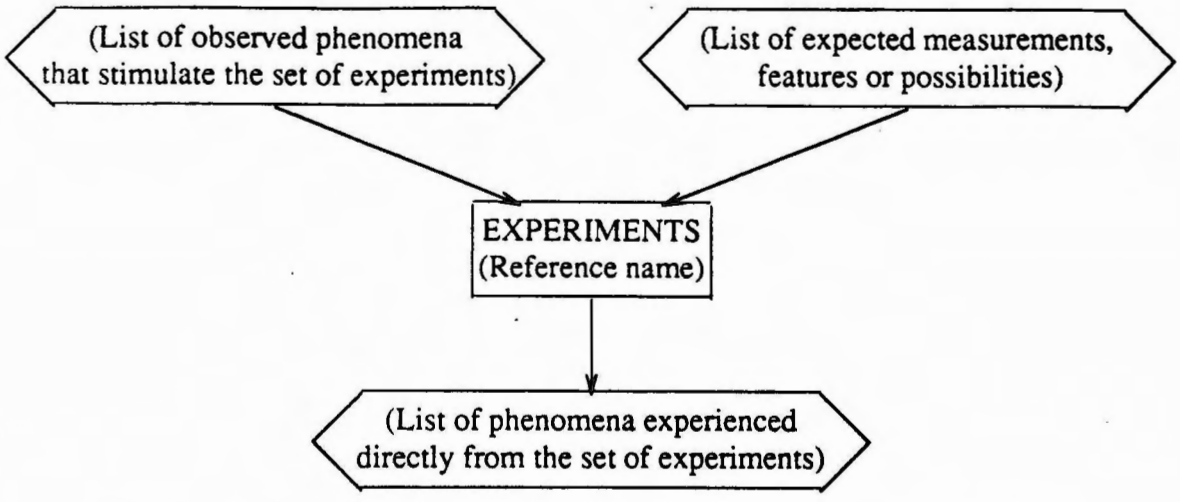


Figure 1. The framework for an extensional description of an experiment.

the set of measurements, features or possibilities that can be derived from the apparatus. Further, the observations are affected by the 'mental set' of the investigator (his view of the world and the way in which he describes things) derived from such factors as his culture, education and goals. The investigator's goals are particularly important since he will be actively 'looking' for something and interpreting what he sees rather than passively accepting the massive amount of data that might impinge on his senses. Figure 1 shows the representatives of this element (the mental set and the apparatus potential) as the second input to the function EXPERIMENTS.

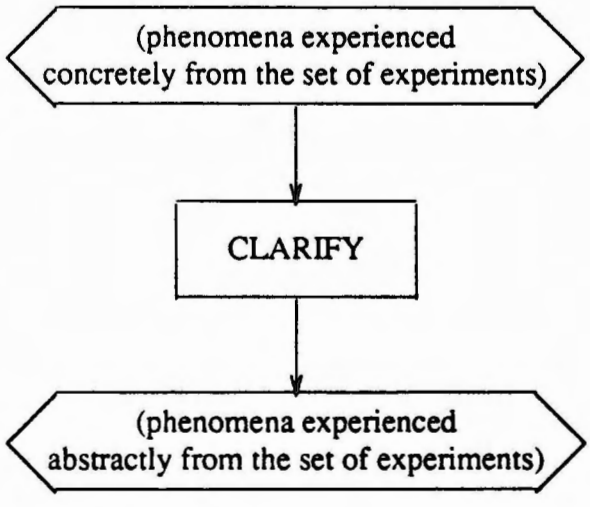


Figure 2. The framework for an extensional description of clarification.

Direct observations of an experiment are not always ideal as an indication of what has occurred. Sometimes the view of the phenomena provides an unclear description of what has happened or it may be that the individual observations represent only components of a single phenomenon. It is an important step to make clear the concrete experiences derived from the set of experiments and recast these experiences into a summary or an abstraction that crystalizes the observations into an unambiguous set of results. Figure 2 shows the schema for the extensional definition of this step that is

embodied in the function CLARIFY. At the level of description presented at this stage the input to CLARIFY will effectively be the same as the output. For this reason it will not be given as part of the analysis; the evidence for this function will be given as extracts from Faraday's notes. However, as will be seen in the formal definition, the change is represented by an alteration of the classification of the phenomena. The input phenomena (before clarification) are in the class:

- Clear or under-specified phenomena that belong to the same class as *observed* phenomena.

Whereas, the output (after clarification) is in the class of:

- Clear or under-specified phenomena that belong to the same class as *abstracted* phenomena.

It might be argued that CLARIFY should perhaps form part of a prior and closely related process, CONSTRUAL; it does, however, form a bridge between the empirical world of observations as identified by EXPERIMENTS and the represented world as identified by CONSTRUAL.

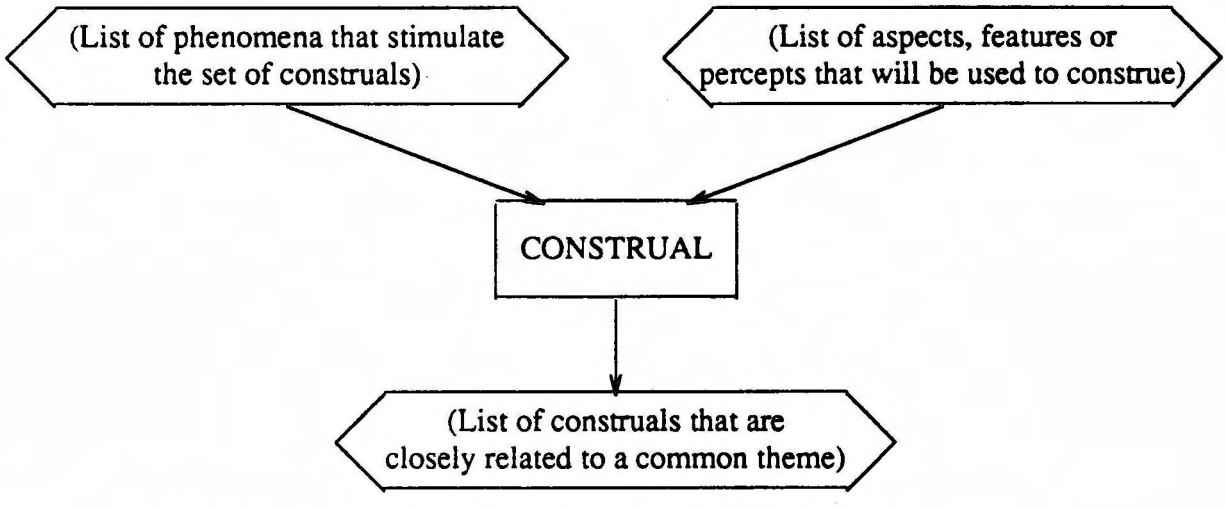


Figure 3. The framework for an extensional description of construal.

Figure 3 shows the schema for the extensional definition of CONSTRUAL. CONSTRUAL is interactive with EXPERIMENTS in that it encompasses the step to propose new phenomena (and possibly new goals). The stimulus for a construal is the list of known phenomena that have been abstracted or represented in a way that leads naturally (and abductively) to a new view (see Gooding 1986, 1990b). This view will still be bounded (as is EXPERIMENTS) by the mental tools available to the investigator. These tools provide the means by which a model of the world is created; they are a list of aspects, features or percepts used to describe the world.

The three extensional functions EXPERIMENTS, CLARIFY and CONSTRUAL are *driven* by an intensional function EXPLORE. EXPLORE models vigorously an informal process of practical reasoning. We describe it in a later section.

3. Creating the Extension of the Model

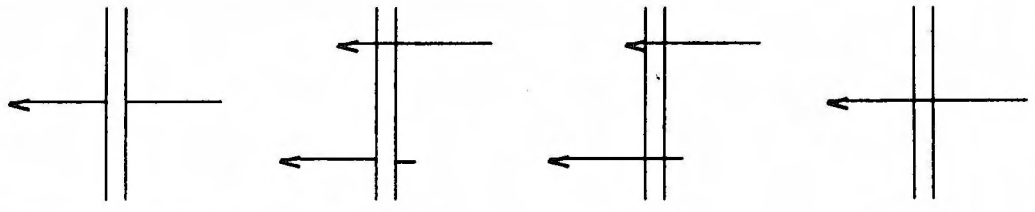
The problem is to find ways of recovering situated exploration processes from accounts that are necessarily reconstructed and propositional. This means recovering more of the experimental situation than historians can usually do, going behind talk and text to look at actions in their material context. In particular, it requires a determined attempt to set aside the organising and simplification effects of hindsight. To recover the contingency and the prearticulate processes, the mental (CONSTRUAL) and the material (EXPERIMENTS) procedures are placed on a par.

Science educators and research scientists know that the creativeness of exploratory experiment owes much to uncertainty. A state of not knowing how to proceed enables the exploration of several possibilities as if each were equally possible or equally feasible. The nontriviality of certain moves or choices emerges only when we remove the obviousness that hindsight has conferred on them. Repeating an experiment can bring out the uncertaintity of observational situations, blurring the clarity of hindsight and so highlighting the creative possibilities available in situations of uncertainty.

EXPERIMENTS-1

Electromagnetic expts. with Hare's Calorimeter. To be remembered that this is a single series.

1. Position of expt. wire A*.
2. Positions at first ascertained were as follows



3. On examining these more minutely found that each pole had 4 positions, 2 of attraction and 2 of repulsion, thus

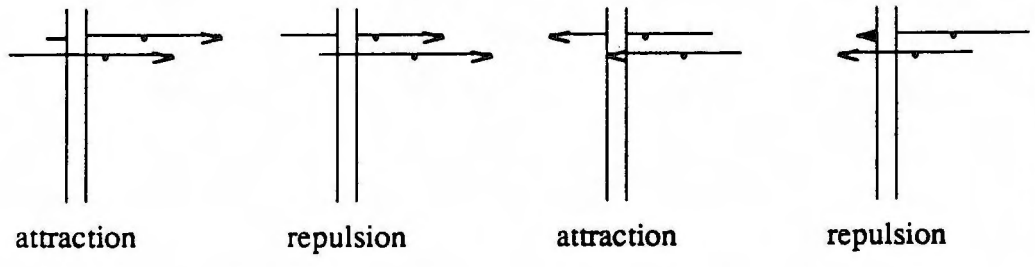


Figure 4. A description of the first experiment from Faraday's notes.

The main source for Faraday's rotation experiments was his laboratory record for his first day's work on this problem (Martin 1932 - 36), supplemented by other contemporary materials, such as laboratory manuals, accounts of earlier experiments he

performed with Sir Humphry Davy and others, and reports of experiments by others interested in electromagnetism.

3.1. The Context of Faraday's Experiments

Figures 4 to 8 are a reproduction of part of four pages of Faraday's notes and the extensional definitions of the functions that represent the sections of these notes. A brief description of the context of these experiments will explain their significance for Faraday.

At the end of the previous summer, news of Oersted's discovery of electromagnetism reached London. Oersted showed that a current-carrying wire affects a magnetised needle. This had an enormous impact becauses it demonstrated a connection between two large but formerly distinct phenomenal domains, electricity and magnetism. Electromagnetism foretold a synthesis of Newtonian proportions. However, Oersted's currents affected his needles in a rather curious fashion, suggesting that the grand synthesis might not be along Newtonian lines. The needles did not behave as they would if they were subject simply to ordinary, Newtonian push-pull forces. There was an apparent tendency to move at right angles to the wire (not simply to or from it). Just as extraordinary was the observation that the action changed from push to a pull, depending on position.

In London, Humphry Davy, W. T. Brande, and others made many experiments on the phenomena. Faraday assisted in some of Davy's experiments in 1820 and 1821. During the summer of 1821, he also reviewed the new field of electromagnetism. By September he was familiar with what others had written and with how far their verbal accounts could be translated into repeatable observations. Faraday found that each investigator produced a different interpretation of the effect, but those lacking first-hand experience could not even visualize the behaviour of a magnetized needle near a wire.

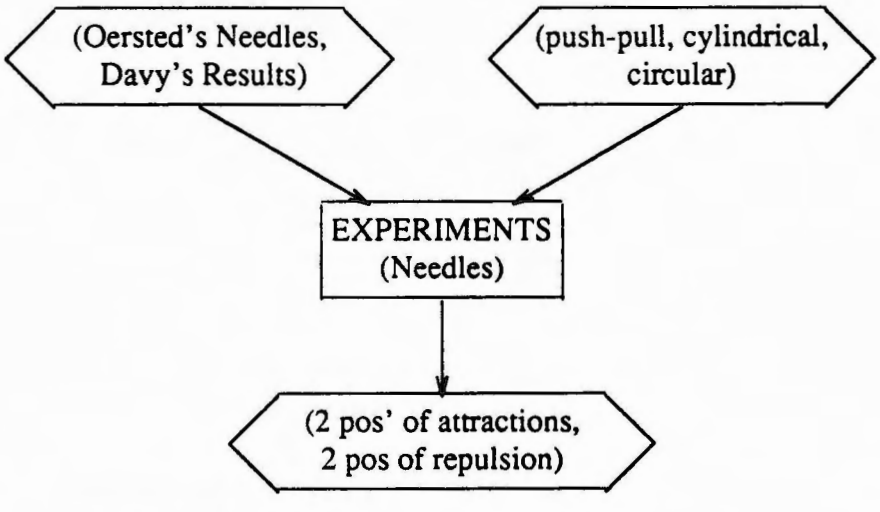


Figure 5. A functional description of the first experiment.

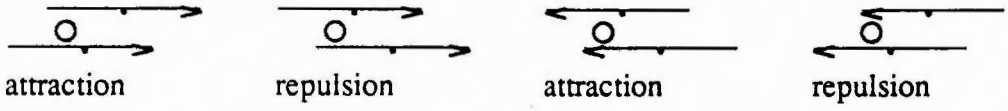
This backgound knowledge gives us the information needed for the extension of the first functional description of the first set of experiments that was carried out on September 3rd 1821. This description is shown in figure 5. The long history of Faraday's experiences with Davy and his knowledge of Oersted's results are indicated briefly by

'Oersted's Needles' and 'Davy's Results'. This abbreviation is all that is necessary since the function is to be described extensionally and any further detail could not be used at this level of representation. Intensional descriptions emerging from further research could be added.

The list of expectations, features or possibilities reflects the conceptualization of hands-on experience of the needle and magnet behaviour at the time (see Gooding 1990b, Chapter 2). The experimental set up would certainly allow these possibilities to be suggested even if (as in the case of circular motion) they are not directly observed.

CLARIFY-1

4. Or looking from above down on to sections of the wire



5. Or

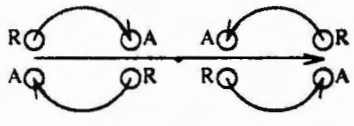


Figure 6. Clarifying the observations of the first experiment.

Figure 6 marks the clarification stage where Faraday summarizes his observations with the needles. This diagram draws together all the effects into two simple descriptions of attraction and repulsion; the diagram prepares the ground for the abductive leap that occurs in the construal stage.

CONSTRUAL-1

6. These indicate motions in circles round each pole, thus



Hence the wire moves in opposite circles round each pole and or the poles move in opposite circles round the wire.

Figure 7. Construal from the diagrams describing the observations.

The diagram in figure 7 gives the first clue that circular or cylindrical movement of the poles or wire should be considered. The choice of a circular behaviour was taken because there is evidence that this image was one of several heuristics for exploratory work by Davy, Faraday, Wollaston, Biot and others, at this time (see Table 1).

1.	Ampere (push-pull forces)	m1	wires move parallel to themselves
2.	Wollaston (axial rotation of magnetism or current)	m2	wires rotate about fixed axis or translate horizontally
3.	Davy (revolving magnetism)	m3	wire rolls (incorporates m2)
4.	Davy (disc/chords)	m4	wire moves perpendicular to magnet
5.	Faraday (positions of attraction/ repulsion)	m5	wire moves continuously (construes needle behaviour through m2 - m4)
6.	Davy (electric arc experiment)	m6	magnetism/current 'revolve' without wire

Table 1. Construals of phenomenal possibilities in play on 3rd September 1821.

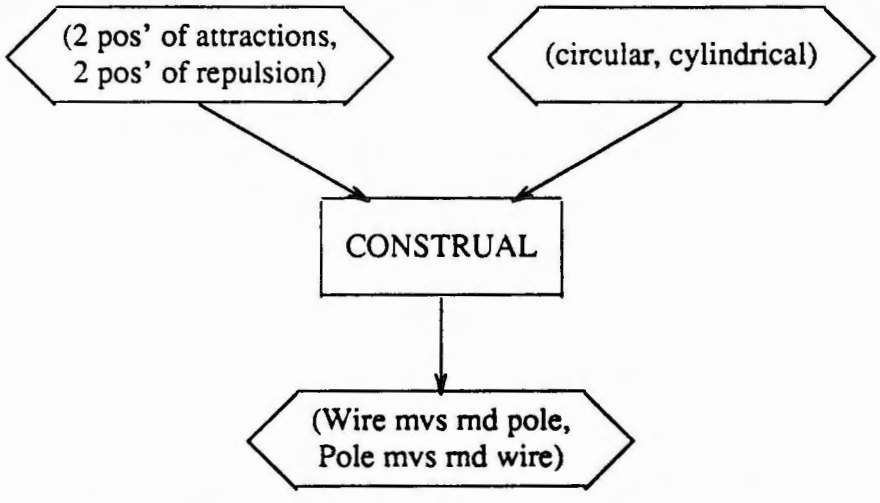


Figure 8. A functional description of the first construal

4. The formation of a Taxonomy

Each component of the extensional functions EXPERIMENTS, CLARIFY and CONSTRUAL has been justified (see Addis et al 1990a for a full account). We have justified them by linking each component with text, historical evidence and the repetition of the experiments. Further components can be added for each of the extensional functions as new historical evidence emerges. However, these components remain dormant until they can be activated through a function that will use the data to recompose the process of discovery and invention.

Before a function of any kind can be exploited by FAITH it has to be *declared.* Declaration is where the function is defined in terms of the types of objects or entities that form its input and its output. The types chosen must provide useful distinctions in the world and by examining the objects or entities we have already chosen for our extensional definitions we should be able to abduce a taxonomy (Addis 1989).

Both the first parameter and the only output parameter of each function are *phenomena* of some kind; in fact a collection of phenonena since each function can accept and generate a list or stream of information.

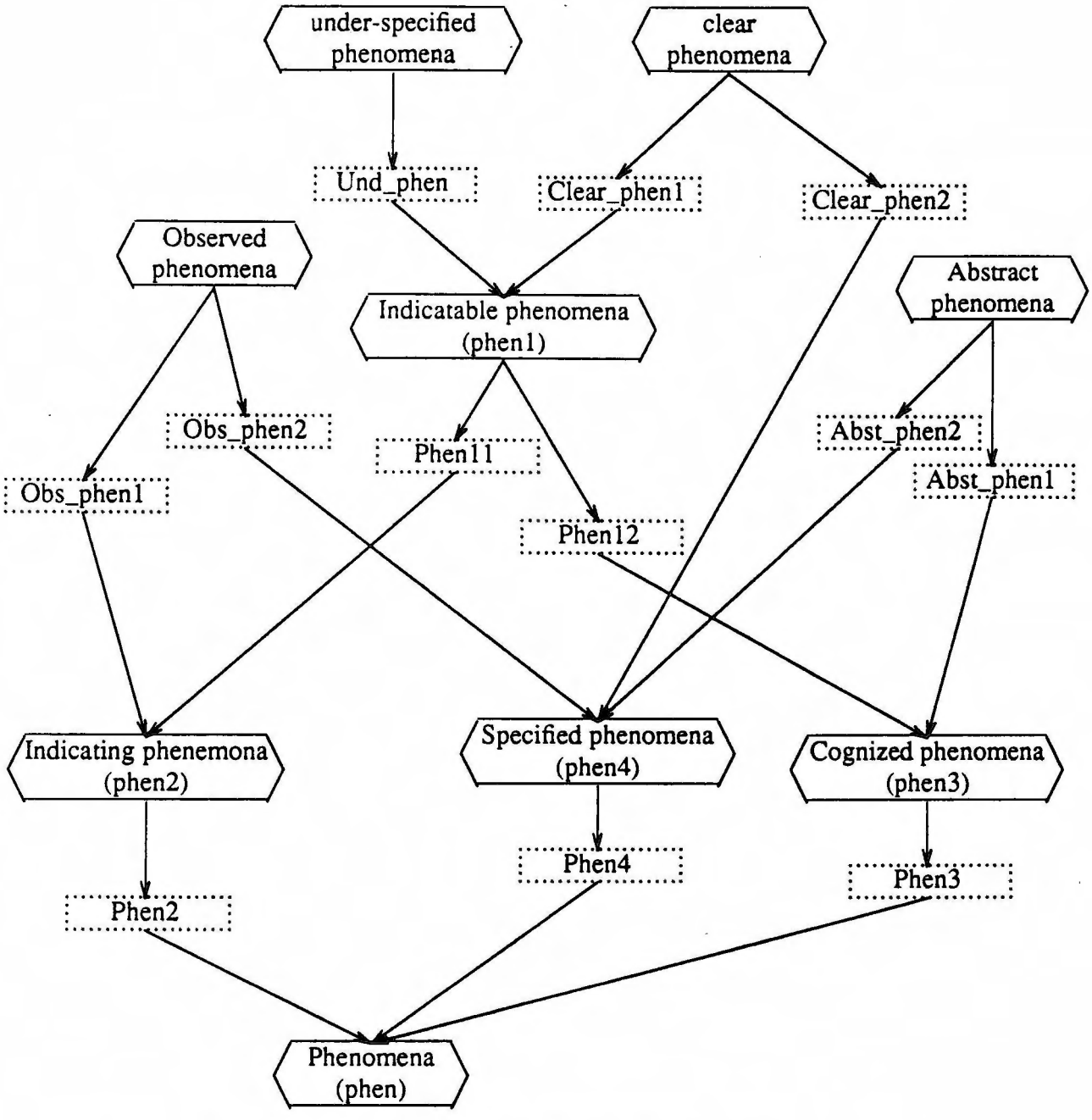


Figure 9. The type structure for the functional model.

However, EXPERIMENTS will accept its first parameter from one of:

- Observed phenomena (viz. Side-ways thrust)
- Under-specified phenomena (viz. Oersted's Needles, Davy's Results)
- Clear phenomena (viz. Wire mvs rnd pole, Pole mvs rnd wire, single pole in centre causes wire to rotate)

and will output one of:

- Under-specified phenomena (viz. 2 pos' of attractions, 2 pos of repulsion, 2 mvmnts of side-to-side, 2 sideways mvmnt of crank)
- Clear phenomena (viz. No Wollaston's revolution, Unassisted circular motion)

CLARIFY will accept for its first parameter one of:

- Under-specified phenomena (viz. 2 pos of attractions, 2 pos of repulsion, 2 sideways mvmnt of crank)
- Clear phenomena (viz. No Wollaston's revolution, Unassisted circular motion)

and will output:

- Clear phenomena (viz. 2 pos' of attractions, 2 pos' of repulsion, 2 mvmnts of side-to-side, 2 sideways mvmnt of crank, Unassisted circular motion)

The same descriptions have been used for both under-specified and clear phenomena here because no 'process' of clarification was recorded. In these cases it is assumed that the observations emerged as clear and better specified although Faraday's description remains the same. More detailed work is needed at this point to adjust or justify our model, as well as work on other examples.

CONSTRUAL on the other hand will accept for its first parameter one of:

- Observed phenomena (viz. 2 mvmnts of side-to-side, 2 sideways mvmnt of crank)
- Clear phenomena (viz. 2 pos of attractions, 2 pos of repulsion)
- Abstracted phenomena (viz. unassisted circular motion)

and will output one of:

- Under-specified phenomena (viz. side-ways thrust)
- Abstracted phenomena (viz. mvmnt of pole causes wire to reciprocate)
- Clear phenomena (viz. wire mvs rnd pole, pole mvs rnd wire, single pole in centre causes wire to rotate).

Again there is a shift in the classification of the phenomena due to the context in which it is acknowledged.

Figure 9 shows the integrated structure of these types (FAITH code is generated from this picture by CLARITY - see Addis and Townsend 1989 and Addis et al 1990a). The intermediate groupings (e.g phen1) reflect the groups defined by the function inputs and outputs. Phen1, for example, is derived from the combination of phenomena that are output from the experimental results. The term *indicatable* for phen1 is used to denote the fact that the phenomena have the potential to indicate observable events. On the other hand, phen2 incorporates phen1 by adding observable phenomena to the class. Phen2 is the input group of phenomena to EXPERIMENTS. We could thus write:

```
fdec
   EXPERIMENTS ::= (list phen2) * (list potentials) -> (list phen1);
;
```

This states that the input parameters of EXPERIMENTS are 'a list of phen2s' and 'a list of potentials' producing 'a list of phen1s'. However, this is not exactly what is written in

practice although the intention is the same†. Similarly, CLARIFY might be written as:

```
fdec
   CLARIFY ::= (list phen1) -> (list clear-phenomena);
;
```

and CONSTRUAL as:

```
fdec
   CONSTRUAL ::= (list phen4) * (list potentials) -> (list phen3);
;
```

where phen4 is the set of specified phenomena (all phenomena except the under-specified), potentials is the set of features available to produce mental models and phen3 is the set of resulting cognized phenomena.

5. The Intensional Function EXPLORE

The list of potentials associated with the functions EXPERIMENTS and CONSTRUAL represent, at one level, the focus of attention of the investigator (Faraday in this case). The shifts and change in the focus of his attention are represented as two lists of list of potentials. This list is an ordered set of collections of potentials; the collections are ordered because it shows a progressive change and the individual items in each collection are assumed to have no particular order (at least for the moment we take this as the case).

We have treated the two lists of list of potentials differently at this stage because we are not sure whether the distinction between the potentials used by EXPERIMENTS and the potentials used by CONSTRUAL is important. Different arrangements of these lists of list of potentials will describe different sequences of thought (as foci of attention) and consequently would suggest different lines of investigation, experimentation and conclusions (see figure 11). Currently, these lists are hand crafted from the historical evidence but alternative scenarios might be explored by creating different sequences. These alternatives can be activated through the function EXPLORE.

Figure 10 shows the recursive function EXPLORE (FAITH code is generated by CLARITY from this picture). Explore links the extensional functions together according to the two lists of list of potentials and the initial input to EXPLORE. In this case the initial condition is *Oersted's Needles and Davy's Results*. The first collection of potentials in the list is *push-pull, cylindrical, circular* (see figure 5). The results of EXPERIMENTS will be *2 pos' of attractions, 2 pos' of repulsion* which through CLARIFY will trigger CONSTRUAL. An extensional definition of CONSTRUAL will respond only if the inputs match. This match will occur if the first (the head) collection of potentials on the list is *circular, cylindrical* (see figure 8). This match will result in the output of CLARITY the list *wire mvs rnd pole, pole mvs rnd wire*. This output will form the new

† The symbol * (called the cross product) indicates that any pair of 'values' (in this case it will be a pair of lists, one of phen2s and the other of potentials) taken from the two parameters are a legitimate input to the function. However, in FAITH the symbol -> is used instead. This makes the description less easy to decipher and consequently for the purposes of this text we resort to the cross product symbol (*).

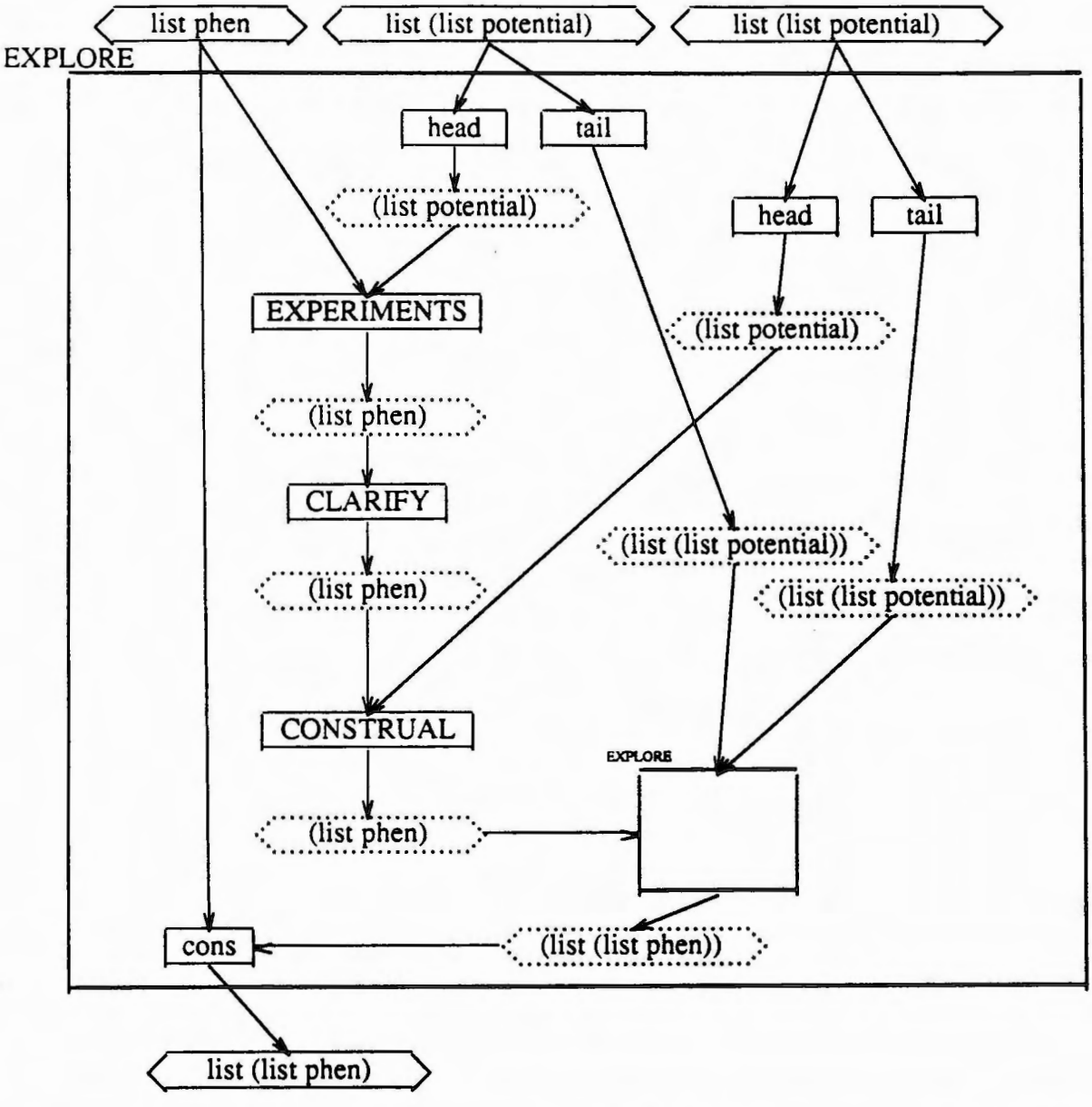


Figure 10. The intensional and recursive function EXPLORE as expressed in CLARITY. starting conditions for EXPLORE in conjunction with the two foreshortened (the tail) lists of list of potentials. Provided these lists are set to pursue a viable sequence of thought the process of EXPLORE will terminate with a trace (a record) of an investigation.

We experimented with two viable lists of list of potentials. The first reproduces the sequence of events that occurred on September 3rd 1821; a sequence that resulted in the rotation motor. The second list reproduces a sequence of events that might have taken place that day resulting in a reciprocating motor. This sequence did eventually take place when Barlow in 1822 created the pendulum motor (see figure 11).

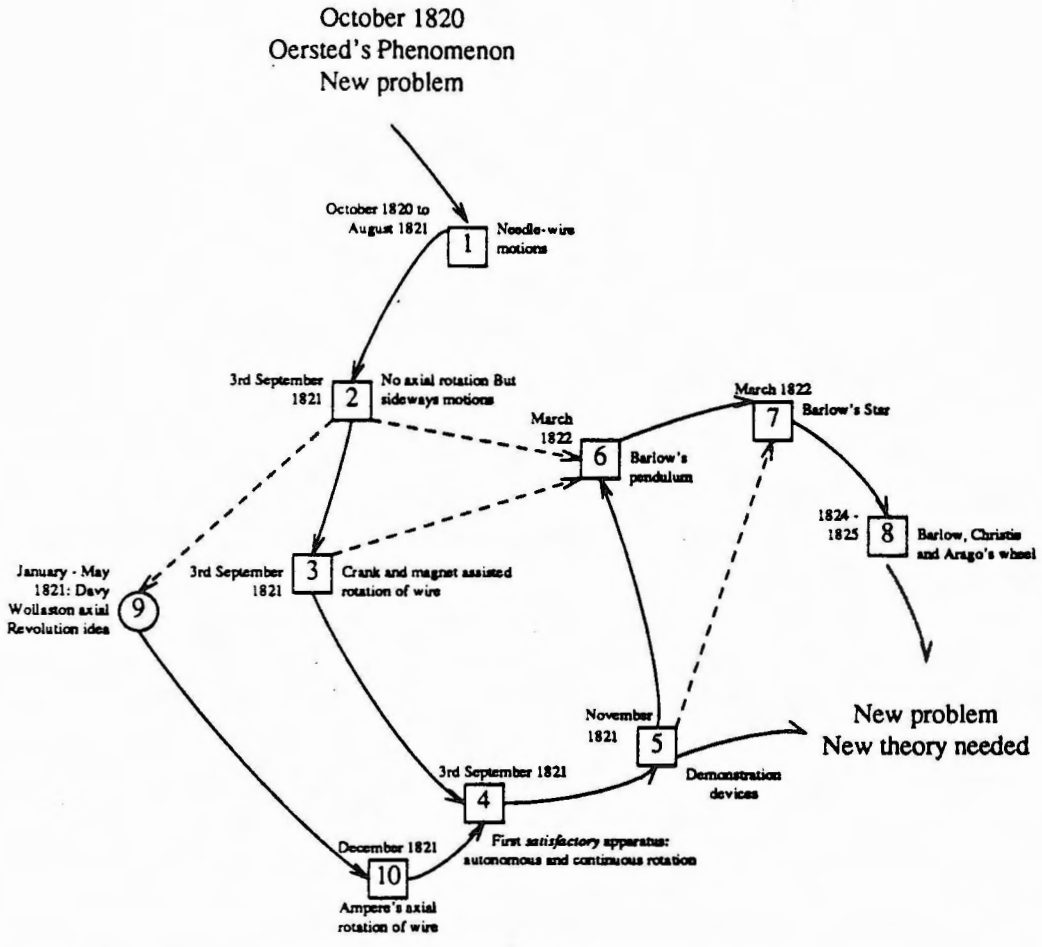


Figure 11. Discovery-paths to several types of electromagnetic rotation.

5.1. Exploring Multiple Discovery Paths

The result of running EXPLORE on the first list produces a sequence of observations. This sequence traces the events that occurred on the 3rd September 1821 and are depicted in figure 11 by the sequence 1, 2, 3 and 4. EXPLORE represents this sequence by a list of lists as follows:

[
 [(Indicating phenomena *Oersted's Needles*) and (Indicating phenomena *Davy's results*)]
 [(Indicating phenomena *wire moves round pole*) and (Indicating phenomena *pole moves round wire*)]
 [(Indicating phenomena *sideways thrust*)]
 [(Indicating phenomena *single pole in centre causes wire to rotate*)]
 [(Indicating Indicatable phenomena *unassisted circular motion)*]
]

The result of running EXPLORE on the second list produces a sequence that picks up Barlow's train of discovery. Figure 11 suggests this train of events by the sequence 1, 2, 3, 6 and 7. EXPLORE represents this possibility of discovery by the list of lists as follows:

[
[(Indicating phenomena *Oersted's Needles*) and (Indicating phenomena *Davey's results*)]
[(Indicating phenomena *wire moves round pole*) and (Indicating phenomena *pole moves round wire*)]
[(Indicating phenomena *sideways thrust*)]
[(Indicating phenomena *movement of pole causes wire to reciprocate*)]
[(Indicating Indicatable phenomena *unassisted reciprocal motion*)]
]

6. Conclusions

It has been shown that the functional database language FAITH can enable the creation of a wide range of models. The range of models extends from those based on observation alone, with very little theoretical content, to those models based on a formal theory which may be induced from observations. The great advantage is that it is possible to move from a qualitative theory and novel phenomena to a rigorous theory based on established facts as and when insight occurs. This move can be done without fixing function types once and for all, and as such it provides an ideal exploratory tool.

The visual program development environment CLARITY provides access to the formal language FAITH for those who are not computer-literate. This is achieved by the natural translation of pictures into the FAITH language and the natural environment for the development of a conceptual framework of empirical data. It is 'natural' because:

- the environment sets no restrictions on the style of development; it allows piecewise development of an idea that is not confined to top down or bottom up constructions.
- the form of the representation can include a mixture of pictures and program; a mixture that depends upon the 'appropriateness' of the form.
- the system takes account of the limitations of human memory by keeping track of function definitions.
- simplifications of the overall structure of the program can be created through a process of filtering and abstraction.

Moreover

- the complexity of a drawn function can always be matched to the ideal level of complexity of a FAITH function. This provides a natural control over the structure of the resulting program.
- it provides a canvas upon which working models of any level of abstraction can be constructed.

CLARITY is still in an early stage of development and more experience with a wide range of users will be necessary to create a finer tool. There is evidence that the CLARITY environment would be popular in schools and in the business sector as well as in scientific and scholarly pursuits. The CLARITY environment would be in the same category of computer programming as the *spread sheet* in that it provides a visual means of access for non-professional programmers.

The functional model that describes Faraday's work has been the first attempt to create a formal system that expresses an *observed* discovery process. We have limited

this initial attempt to one (important) day in the work of Faraday. It is expected that the model will develop to take into account the experiences of the investigator studied (here, Faraday) over a much longer period. We would hope to create a model that would describe *the research* activity of several eminent scientists. As we learn more from the laboratory notes and other sources it should become clear what it is that controls the changing focus of a particular scientist's attention as he explores via the processes of experimentation and thought. These controls would then be modelled as further functions either extensionaly or intensionally. The preference is always to create models with a minimum of extensionally defined functions so as to encourage the production of a rigorous theory.

What is not clear from these results is the mechanism through which genuinely new features, attributes, distinctions and potentials can occur. Although Faraday's day's work ended with a new phenomenon, in the limited experience of our investigation there are no examples of a new feature being introduced. In the model hindsight helped us to steer a course through possible sequences of events (Gooding, 1989). New views of the world are certainly formed by the original investigator and we have illustrated how that process occurs. The functional tool therefore opens up new ways of representing historical processes; ways that will enable scholars to model human behaviour in a new framework.

7. REFERENCES

Addis T. R. (1985) *Designing Knowledge-Based Systems*, pub by Kogan Page, distributed by Chapman and Hall.

Addis T. R. (1989) *The Science of Knowledge: A research programme for knowledge engineering*, 3rd European Workshop on Knowledge Acquisition for Knowledge-Based Systems, Paris, July, pp 49 - 59. ISBN 2-90367763-8.

Addis T. R. and Townsend J. J.† (1989) *A Reactive Graph Analysis System: Provisional Results*, University of Reading, Computer Science Dept. Technical Report KSG/CARDS/WN/2, February.

Addis T. R., Gooding D. and Townsend J. J. (1990a) *A functional description of Faraday's discovery of the electric motor.* Knowledge Systems Group Report, University of Reading. December. pp1-41.

Addis T. R., Townsend J. J. and Nowell M. C. C. (1990b) *FAITH and CLARITY manual for CARDS*, University of Reading, Computer Science Dept Technical Report KSG/CARDS/13, April.

Gooding D. (1986) *How do scientists reach agreement about novel observations?*, Studies in History and Philosophy of Science, 17, pp 205 - 279.

Gooding D. (1989) "History in the Laboratory: can we tell what really went on?" in F. James, Ed., *The Development of the Laboratory*, Macmillan.

† Townsend J. J. also writes under the name of Loader J. J.

Gooding D. (1990a) "Mapping Experiment as a Learning Process", *Science Technology and Human Values* Volume 15, pp 165 - 201.

Gooding D. (1990b) *Experiment and the Making of Meaning*, pub Kluwer Academic, ISBN 0-7923-0719-4.

Martin T. (ed) 1932 - 36 *Faraday's Diary*. 7 Vols & Index, Bell, London.

Read C. (1989) *Elements of Functional Programming*, Addison-Wesley.

Langley P. (1988) *Machine learning as an experimental science*, Machine Learning 3. pp 5-8.

Structuring Knowledge using Knowledge Graphs

P. James*
c/o H. van den Berg
Department of Applied Mathematics
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands
Email: hvdberg@math.utwente.nl

May, 1991

Abstract

In this paper, a method is given for structuring knowledge acquired from scientific texts. This is done using a formalism called knowledge graphs, and the graph-theoretical notion of construct. An example of this structuring procedure is given, and some further developments are sketched.

1 Introduction

In the last decade, knowledge representation and knowledge acquisition have become important topics of research in Artificial Intelligence, especially for

*The author's name P. James is a pseudonym for the group of researchers that have been working in the project "Knowledge Graphs": R.R. Bakker, H. van den Berg, C. Hoede, M.A.W. Houtsma, H.J. Smit, F.N. Stokman, P.H. de Vries and M. Willems.

the development of expert systems. In many expert systems, the represented knowledge consists of rules of thumb and heuristics. In others, the foundations of these rules, i.e. scientific theories, are represented.

Knowledge acquisition for expert systems is often based on an interrogation of an expert. It turns out that experts have difficulties with the formulation of their strategies and their knowledge used to solve problems [3]. Scientific theories are described in articles and books. Although methods are being developed for knowledge acquisition from experts [6], it might be better to use scientific texts to construct a declarative knowledge base for an expert system.

The aim of the *knowledge graph project* was to develop techniques for this construction of a declarative knowledge base from a set of scientific texts describing a scientific theory. In order to represent such a theory, the knowledge in the documents will have to be extracted, represented, and integrated. An important aspect of integration is to join existing concepts into new ones. We will call this process *knowledge structuring*. For this purpose, a knowledge representation formalism called *knowledge graphs* was defined.

A knowledge graph is a graph in which nodes represent concepts, and arcs represent relations between those concepts. So knowledge graphs are similar to semantic networks and to conceptual graphs, see Sowa [10]. Typically, only a few types of relations are permitted in a knowledge graph, namely those types characteristic for the part of science which is studied. In medical or social sciences, important relations are a causal relation, an aggregation relation and a generalisation relation.

Of course, limiting the number of types of relations restricts the amount of knowledge one can represent in a knowledge graph. But, when one is aiming at structuring knowledge, this loss of detail is not a disadvantage. Structuring means getting rid of detail and focusing on the important parts of the knowledge. Besides, using only a few types of relations facilitates the use of graph-theoretical techniques that can be applied in order to structure the knowledge in that graph.

As said before, the basis for knowledge acquisition in knowledge graphs is documents. The plural "documents" is essential here, because theories, for instance in the medical or social sciences, are in continuous development, and a scientific article usually describes some new aspects of a theory. Therefore it is important not only to extract and represent knowledge, but also to

integrate it with existing knowledge graphs.

Several techniques have been implemented in the program KISS (Knowledge Integrating and Structuring System). The developed methods have been tested using different knowledge graphs in the domains of social dentistry and lung diseases. De Vries described the results of the application of these methods and techniques [11].

In this paper, we will emphasise the detection of coherent parts in a knowledge graph. Finding which concepts belong together is an important aspect of knowledge structuring. For this reason, Bakker developed the mathematical notion of *construct* [1].

In the rest of this paper, first the definition of a knowledge graph is given (Section 2). In Section 3, the theory of constructs is given, and this is applied in Section 4 to a case-study concerning the sulphur cycle in the Ems estuary. We conclude this paper by giving a brief overview of the current research and further developments in the knowledge graph project.

2 Knowledge Graphs

In this section we will describe in more detail the process of knowledge structuring. This process is composed of the procedures text analysis, concept identification, concept integration and link integration. For these procedures we use a formalism called knowledge graphs, that first will be defined. For an extensive description, see [1] or [8].

A knowledge graph consists of labeled points and labeled arcs (directed and undirected). Points represent concepts and arcs represent relations between concepts. A knowledge graph can thus be seen as a model of a certain piece of knowledge. The number of types of relations in a knowledge graph is chosen to be small, because knowledge graphs are used for structuring.

Definition. A *knowledge graph in conceptual form* is a pair $K = (C, R)$ with C the set of concepts and R the set of relations. The relations in R are denoted by triples $c_1 \rightarrow r \rightarrow c_2$, $c_1, c_2 \in C$ and $r \in T$, where T is called the *universe of relations*. r is called the *type* of the relation. For symmetric relations both triples $c_1 \rightarrow r \rightarrow c_2$ and $c_2 \rightarrow r \rightarrow c_1$ exists, and the notation $c_1 - r - c_2$ is used.

The set T of relation types can vary, depending on the domain of knowledge under consideration. In a certain domain of knowledge, those relations are chosen that are characteristic for that domain. (In the case-study in Section 4 we only use a causal and a part-of relation.)

We will next describe the construction of knowledge graphs from scientific documents. The first stage in this construction is *text analysis*. In a document, one distinguishes *operators* and *arguments* in a sentence. An argument is a self-contained word or group of words, whereas an operator is a word or group of words that is dependent on one or more arguments. The arguments are viewed as concepts, and the operators one finds are mapped on relations from T, using a kind of dictionary. In this way, a knowledge graph is built from a text.

For example, in the sentence "asthma causes emphysema", *asthma* and *emphysema* can be seen as arguments and *causes* as an operator, that can be mapped on a causal relation in T. This method of text analysis has been described by De Vries [11].

Using this technique, every document results in a knowledge graph. To achieve structure of the knowledge in these graphs, the graphs have to be merged and integrated into one graph. The first step in this structuring process is the integration of the different terminologies used in the documents. Identical or similar concepts (synonyms) in the knowledge graphs have to be recognised and identified. This process is called *concept identification* and is based on *similarity measures*. These measures (see [7, 9]) calculate the resemblance of concepts based on the comparison of the names and the structural positions of concepts in the graphs. Weight factors in the similarity measures can be varied, emphasising either the similarity in names or the structural positions in the graphs. The graph, resulting from this procedure, is called the *compiled graph*.

This compiled graph may still contain global and detailed descriptions of identical phenomena and equivalent descriptions of the same phenomena in subgraphs. A procedure called *concept integration* introduces *frameworks* that represent coherent parts of knowledge, and identifies similar parts of knowledge. This is done using so-called *constructs*. This will be elaborated in the next section. The use of these frameworks makes it also possible to consider the represented knowledge at different levels of abstraction.

Typical questions in expert systems are concerned with making inferences, for instance the question of explanation of certain observed phenomena. By

describing how different relations in a knowledge graph can be combined with each other, this can be achieved. A so-called *path algebra* can be defined, that defines the interaction of different types of relations. This procedure is called *link integration*. The knowledge graph, resulting from this last step, is called the *integrated graph*, and can be used as a knowledge base for an expert system. It represents a part of scientific knowledge.

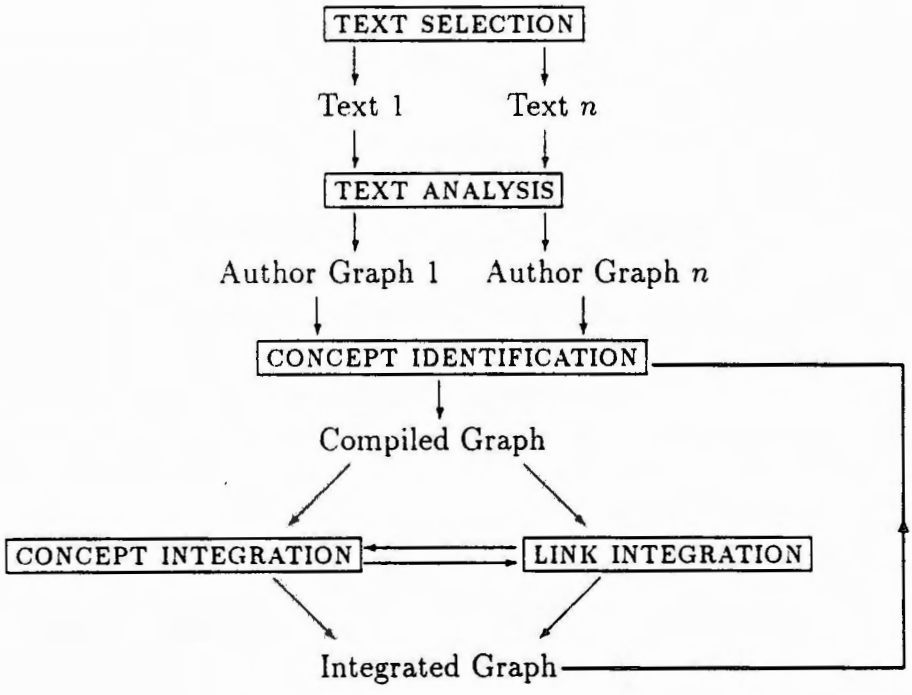


Figure 1: Schema KISS

The interaction of the procedures described above can be shown schematically as in Figure 1.

3 Constructs

In the preceding section, an overview was given over the process of knowledge structuring in knowledge graphs. We will now focus on one element of this

process, namely the finding of coherent parts of knowledge, by which the structure of knowledge will become more transparent.

The goal of defining constructs was to find coherent parts of a graph, i.e. a group of points that have a similar position with respect to the rest of the graph. Constructs form structural units in a graph. Which structural units can one distinguish in a graph?

Obvious candidates for such units are single points and the whole graph, because on some other level of abstraction points may represent a whole theory or a theory may be summarised by a single point.

Every weakly connected component of a graph is also a structural unit. This is reasonable, because a weakly connected component denotes a group within a graph (see [5]).

Another important structure which can be classified as a structural unit is the strongly connected component in a graph. To see this, suppose a graph represents a causal net, points representing concepts and arcs between points representing causal relations. An important notion in these causal nets is a process with feedback, where from any point in that process any other point in the process can be reached by a directed path. In graph-theoretical terms this means that this process forms a strongly connected component.

A construct, as defined below, generalises these four structures [1]. It is an induced subgraph of a knowledge graph that forms a coherent part of the knowledge graph with respect to the rest of the graph. It can represent a coherent fragment of knowledge, a process with feedback, or a set of similar fragments of knowledge.

Bakker defined constructs for directed graphs and gave algorithms for the detection of constructs in acyclic directed graphs [1]. Willems restricted these definitions so that algorithms could be applied to arbitrary directed graphs [12]. In this paper we will restrict ourselves to the definition of Bakker [1].

In the following graph-theoretical definitions, the terminology of Harary [4] is used. Labeled directed graphs with point set P and arc set A will be denoted by $D(P, A)$. An induced subgraph S of $D(P, A)$ with point set Ps will be denoted by $S(Ps, As)$ or $< Ps >$. A directed path π in a directed graph $D(P, A)$ is a sequence $p_0, a_1, p_1, a_2, \ldots, a_n, p_n$ with $p_0 \in P$, $a_i = (p_{i-1}, p_i) \in A$, $p_i \in P$ $(i = 1, 2, \ldots, n)$, in which all points, except possibly p_0 and p_n, are distinct.

Definition. An *in-neighbour*, respectively *out-neighbour, of a subgraph*

$S = < Ps >$ in a directed graph $D(P, A)$ is a point $p \in P - Ps$ for which there exists an arc $(p, p_s) \in A$, respectively $(p_s, p) \in A$, with $p_s \in Ps$. The set of in-neighbours respectively out-neighbours of a subgraph $< S >$ is denoted by $In(S)$ respectively $Out(S)$.

Definition. For an induced subgraph $< S >$ of a directed graph $D(P, A)$ a *shared in-neighbour* respectively a *shared out-neighbour* is a point $p \in P - S$ such that for all $s \in S$ there exists a directed path π from p to s respectively from s to p, such that all points in π except p are in S. The set of shared in-neighbours respectively shared out-neighbours of a subgraph $< S >$ is denoted by $SharedIn(S)$ respectively $SharedOut(S)$.

Definition. An *in-construct* respectively *out-construct* in a directed graph $D(P, A)$ is an induced subgraph $< C >$ such that $In(C) = SharedIn(C)$ respectively $Out(C) = SharedOut(C)$. A *construct* is an induced subgraph $< C >$ that is an in-construct and an out-construct. We shall say C is a construct.

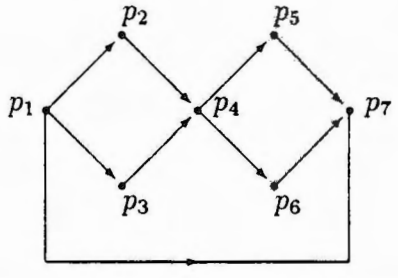


Figure 2: An example of constructs

For example, in Figure 2, a small graph is given. The induced subgraph $< p_2, p_3, p_4 >$ is a construct. The point p_1 is a shared in-neighbour and the point p_6 a shared out-neighbour of this subgraph. KISS has the capacity to find constructs and in- and out-constructs.

Usually, most subgraphs of a directed graph will not be constructs. The number of constructs can, however, still be very large. Some constructs are less interesting, e.g. constructs consisting of a single point, and constructs that can be extended with a point such that the extended subgraph is a construct too. Therefore we introduce the notion of *maximal construct*.

Definition. A *maximal construct* $S =< Ps >$ in a directed graph $D(P, A)$ is a construct with the property that no point $p \in P - Ps$ exists such that $< Ps \cup \{p\} >$ is a construct.

It can be proven that the non-empty intersection of two maximal constructs is again a maximal construct [1].

These definitions can be applied to directed graphs. However, a knowledge graph in conceptual form, as defined in Section 2, is not a directed graph. It is straightforward to translate a knowledge graph in conceptual form to an equivalent directed graph [4]. We can apply construct analysis to that graph.

Although the notion of construct was defined to detect structure in causal nets, it can be used for other relations as well. However, further research is needed to determine the exact meaning of a construct in a graph with other relations. When using the system KISS to determine constructs in a graph, the user is asked to decide whether or not a construct makes sense to him. In this way we can prevent adding meaningless constructs; however this does not often occur.

In the next section, an example is given to show the strength of constructs.

4 The Ems Estuary

In this section we will present an example of the use of constructs in the structuring of knowledge using knowledge graphs. The following is described in detail by Smit [9].

In [2], an attempt was made to construct a knowledge graph of the Ems estuary, a river bordering both Germany and Holland. A part of this graph, describing the sulphur cycle in the estuary, was analysed further. On the basis of a scientific article the knowledge graph depicted in Figure 3 was constructed. In this graph, the universe T of relation types consisted of two types, a CAU-relation (depicted by $\rightarrowtail$) and a PAR-relation (depicted by $\rightarrowtail\!\!\!\!\circ$). A CAU-relation between a and b is a causal relation, expressing that "a change in or occurrence of a can cause a change in or occurrence of b". A PAR-relation is a part-of relation.

By using KISS, construct analysis was applied to this graph to find the maximal constructs in this graph. The results are denoted by marked frames in Figure 3. The points in such a frame are contracted into one point.

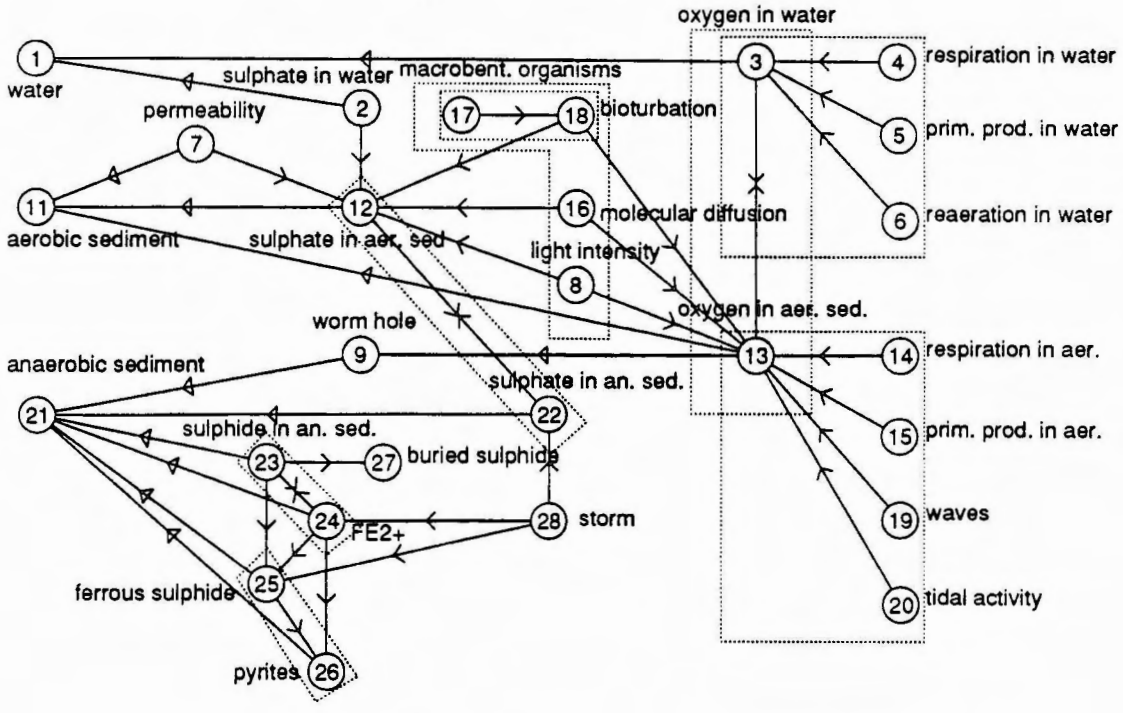


Figure 3: A knowledge graph of the sulphur cycle

To make the structure of this graph more explicit, *level assignment* was applied. Level assignment assigns a level to each concept, depending on the number of incoming and outgoing arcs (see [4]). The resulting graph is drawn in Figure 4.

Seeing the structure of this graph, we can observe the following. It seems as if some concepts are missing in this model of the sulphur cycle, namely the dotted concepts a, b, and c. For example, on the left side we have the concepts 1 ("water"), 11 ("aerobic sediment"), and 21 ("anaerobic sediment"). A PAR-relation from concept 27 ("buried sulphide") to a possible concept "underground sediment" might be missing, due to the fact that this graph stems from some larger model. After consulting the authors, it turned out that this was indeed the case. Even the name of the missing concept was right.

This example shows that construct analysis leads to a simpler and transparent structure, representing a model of the knowledge in an article.

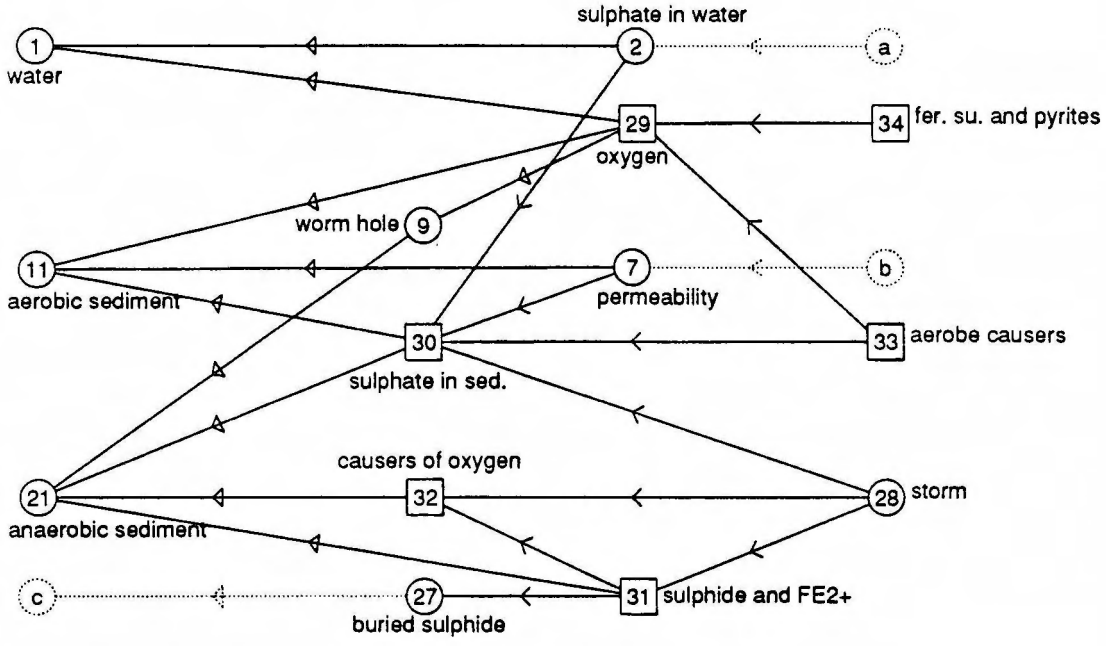


Figure 4: A knowledge graph of the sulphur cycle after construct analysis

5 Further Research

In the preceding sections, we have indicated how one could structure knowledge using knowledge graphs and how one could construct a qualitative model of some piece of knowledge. However, we gradually have become aware of the fact that knowledge graphs can be used for more general objectives like knowledge representation and natural language representation. So the emphasise in the knowledge graph project has shifted from knowledge structuring to knowledge representation.

Knowledge structuring, as described in this paper, can be seen as a kind of "loose" modelling, where different operators in a text are mapped on one common, underlying relation in knowledge graphs. For instance, a complex relationship is said to be of type CAU because it contains a causal aspect. When the emphasise in the project shifted towards knowledge representation, "loose" modelling became "sharp" modelling, where a relation is considered to be of a certain type only in very restricted cases. Our current research investigates the consequences of this assumption.

A first question we have addressed is the number and types of relations. A few relation types are viewed as primitive, and we assumed that, on deeper reflection, more complex relations show a structure in terms of those few types. A common feature of semantic networks is the great number of different relation types. The fact that there is such a large diversity within the field of semantic networks regarding the types of relations, indicates that the choice of relation types in most semantic networks is somewhat arbitrary. We try to find a small basis of relation types for all the relations used in semantic networks.

Another question we have addressed is the use of knowledge graphs for the representation of the meaning of words in natural language. We assume that every word has a meaning, which can be represented by a (small) knowledge graph. The meaning of a sentence (or of a larger piece of discourse) can be found by combining the knowledge graphs of the words occurring in that sentence. The process of combining these graphs is analogous to combining knowledge graphs as described in the foregoing sections. Similarity measures are used to make out which graphs can be joined. The result is a larger graph representing the meaning of a sentence.

6 Conclusions

We have given a method to extract knowledge from (scientific) documents. Just extracting knowledge is insufficient for practical use of this knowledge. One has to structure that knowledge in order to use it. For this, we developed methods.

We introduced knowledge graphs and defined the notion of construct. A construct is a coherent part of a knowledge graph, for instance representing a set of similar fragments of knowledge, or a process with feedback. It generalises some notions of structure in a graph. Using these constructs, the structure in the graph becomes more transparent. An example was given to illustrate this.

References

[1] R. R. Bakker. *Knowledge Graphs: representation and structuring of scientific knowledge.* PhD thesis, University of Twente, Enschede, 1987.

[2] R. G. Edens and J. Baretta-Bekker. De toepasbaarheid van kennisgrafen als voorstudie bij ecosysteemmodellering. NIOZ Rapport 1989 – 7, Texel, The Netherlands, 1989.

[3] E. A. Feigenbaum and P. McCorduck. *The Fifth Generation.* Addison-Wesley, Reading, Mass., 1983.

[4] F. Harary. *Graph Theory.* Addison-Wesley, Reading, 1972. Third Printing.

[5] F. Harary, R. Z. Norman, and D. Cartwright. *Structural Models: An Introduction to the Theory of Directed Graphs.* John Wiley & Sons, Inc., 1965.

[6] S. A. Hayward, B. J. Wielinga, and J. A. Breuker. Structured analysis of knowledge. In Boose and Gaines, editors, *Proceedings of Knowledge Acquisition for Knowledge-based Systems Workshop.* Academic Press, 1987.

[7] C. Hoede. Similarity in knowledge graphs. Memorandum no. 550, University of Twente, Enschede, 1986.

[8] P. James. Knowledge graphs. Memorandum no. 945, University of Twente, Enschede, 1991.

[9] H. J. Smit. *Consistency and Robustness of Knowledge Graphs.* PhD thesis, University of Twente, Enschede, 1991. In preparation.

[10] J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine.* Addison–Wesley, Reading, 1984.

[11] P. H. de Vries. *Representation of Scientific Texts in Knowledge Graphs.* PhD thesis, Rijksuniversiteit Groningen, 1989.

[12] M. Willems. Construct analysis in knowledge graphs. Memorandum no. 815, University of Twente, Enschede, 1989.

Fifth European Knowledge Acquisition for Knowledge-Based Systems Workshop

EKAW91

Knowledge Elicitation Techniques

Logical methods in protocol analysis

Izak van Langevelde
Jan Treur

Vrije Universiteit Amsterdam
Department of Mathematics and Computer Science
Artificial Intelligence Section
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
email: iavlang@cs.vu.nl, treur@cs.vu.nl

Abstract

Complex reasoning tasks can be analysed by means of a task model that describes how the task can be decomposed into a number of primitive subtasks. We describe a framework that makes a formal specification of a task model possible and which uses standard logics at the level of the primitive subtasks. The use of this framework will be illustrated by means of two examples on hypothetical reasoning. Empirical work on the second example, the wise men's puzzle, is used to demonstrate the use of our formal framework in protocol analysis.

1 Introduction

In this paper we will show the use of logical methods to analyse complex reasoning patterns that occur in (expert) protocols. Of course we are aware of the fact that a straightforward use of a standard logic to describe a whole complex reasoning pattern often does not work. In particular, dynamic events in reasoning processes are not covered by the use of standard logic. Such dynamic events often have a reflective nature, for instance:

- switching from one interpretation or fragment of the world to another one
- assuming a fact for a while and later on withdrawing it
- controlling or focussing the reasoning process

These types of dynamic events cannot be described adequately within one global (standard) logic. Therefore, in the literature on the analysis of verbal protocols (for

instance, see [1], [2], [3], [4], [5], [6]) it is not often advocated that logic should be used as a descriptive framework. Instead, much stress is put on the use of a task model to partition the reasoning task into a number of subtasks. In general, the semantics of such a task model has a procedural nature and is left informal: it models a process decomposition of the reasoning pattern. This leaves open the question whether or not more formal descriptions are possible that have declarative semantics. Our approach uses a framework for such descriptions. We model complex reasoning patterns by combining the use of a (global) task model with the use of logical methods at a local level. We show that at both levels (declarative) formal descriptions can be used.

In this paper, in section 2 we will describe our approach and compare it to other approaches known from literature. In section 3 and 4 two toy example reasoning patterns are analysed. In these toy examples we show how hypothetical reasoning can be modelled by handling dynamic assumptions (introducing and withdrawing them during the reasoning process) in an explicit manner. This provides a basis for section 5, where empirical work is compared and discussed.

2 Description of our approach

Reasoning tasks are described at two levels: the global (task) level and the local (subtask) level. At the global level the task is described by a functional decomposition into its (primitive) subtasks, and by describing the interactions between the subtasks: both the connections between the data that are used and the control flow. At the local level we describe each of the subtasks in more detail: the incoming information (input facts), the outcoming information (output facts), and the knowledge involved in the reasoning of the subtask (specifying logical relations between input facts and output facts). Such a description at the local level defines the *functionality* of the subtask.

According to our view, both locally and globally a complex reasoning process can be described by means of logic. Thus the analysis of a verbal protocol involves the following:

1. *Creating a task model*
Create or choose a suitable task decomposition. This task model provides a fine-grained functional decomposition of the task into primitive subtasks that reason about one homogeneous fragment of the world. This involves a very sensitive identification of the differences between world fragments that are reasoned about during different parts of the task. It is assumed that the primitive subtasks are simple enough to make the following possible.
2. *Making local logical descriptions*
For each of the primitive subtasks, establish its incoming information (starting points of the reasoning in this subtask) and outcoming information (conclusions of the subtask), and make a description of the related knowledge and primitive reasoning process by means of logic. The fine-grained decomposition given by the task model should guarantee that each of the primitive subtasks can be described as a standard monotonic reasoning process.

3. *Relating the local descriptions to each other*
Relate all logical descriptions thus obtained to each other, both in the sense of information (connecting output facts of one subtask to the required input of another one) and control flow (control knowledge: which subtask should be carried out in a given state of the reasoning process).

These three main ingredients all can be given precise and formal definitions; see [7] (step 1), [8], [9] (step 2) and [7] (step 3).
For shortness, sometimes we will omit the word "primitive" from "primitive subtask".

2.1 Descriptions at the local level

We use a logical language to make such a local description (many-sorted predicate logic). The truth and falsity of input and output facts are described as truth values of atomic statements and the knowledge base is described using the normal form given by the rule-format

$$a_1 \wedge \ldots \wedge a_n \rightarrow b$$

Here each of the a_i and b is a literal, that is an atomic statement or a negation of an atomic statement. We have proved that *any* possible primitive subtask functionality (determined by some description of the input-output connection) can be described by a knowledge base in rule-format (if, for instance, the inference mechanism of chaining is used). Therefore by this rule-format no essential restriction is introduced, compared to predicate logic. Since most knowledge representation languages (for instance object or network languages) can be translated into (a subset of) predicate logic, essentially our format is at least as strong as these languages.

2.2 Descriptions at the global level

Essentially the global level concerns the description of a generic task model (as defined in [7]). There are two parts of such a description. The first part describes what data are transferred from one subtask to another one.

In our approach each subtask in the reasoning process is supposed to reason about a specific fragment of (the state of) the world: a world situation fragment. Therefore it deals with a particular partial description of a fragment of that world situation fragment. Here some of these fragments of the world state may overlap and that is the point where data-exchange between these subtasks may enter the scene. In our approach such a data-exchange is established by specifying *transformations* that link atoms of different subtasks to each other, and that describe the semantical connection between the atoms that are linked in terms of the relation between the truth values of them. For example, if **a** is the atom denoting that a patient has fever and **b** denotes that the patient is without fever, then **a** and **b** can be identified with each other, with a semantical relation given by the truth table **<<true, false>, <false, true>>**, i.e. if **a** is true then **b** is false and if **a** is false then **b** is true (and conversely). We will denote this by

$$\neg a \Leftrightarrow b$$

For convenience, we will not discuss here the role of truth values **unknown**.

Of course, in most descriptions of complex reasoning tasks one will use the same names for atoms with essentially the same meaning; in that case the identification as sketched is just syntactical equality. However, nontrivial semantical identifications may be needed if *reflective interactions* take place. Many complex reasoning patterns essentially are reflective (for instance, see [10], [11]). This means that during the reasoning sometimes the subject of the reasoning of a subtask A is not a fragment of the external world but a fragment of the state of another subtask B (i.e. of the reasoning process itself). If a subtask A interacts with another subtask B in this manner, we say that A is a meta-level subtask and B is an object level subtask with respect to this interaction.

In our approach reflective patterns are modelled in a quite integrated manner. If a subtask reasons about some fragment of the system then this is treated similar to reasoning about a fragment of a (changing) world situation. We refer the reader to [7] and [10] for a more detailed introduction to reflective interactions and the role they play in complex reasoning patterns. Two types of reflective interactions can be distinguished: upward transformations (transform information on the current process state of an object level subtask to input information of a meta-level subtask) and downward transformations (transform output information of a meta-level subtask to a process state of an object-level subtask). Within these types of transformations more distinctions can be made and standard (generic) interaction types can be defined (see [12]).

In this paper we will only use some types of these reflective transformations. The only upward transformation we use here is the *epistemic upward transformation*. This transforms (meta-)information about the current information state of the concerning object level subtask to input facts of the meta-level subtask that is concerned. So if an atom **a** of the object level subtask has truth value **true** or **false**, the truth value **true** is assigned to the meta-atom **known(a)**; else the truth value **false** is asssigned. The unary predicate **known** is a standard generic meta-predicate that can be used in the reasoning of the meta-level subtask that is concerned. Similarly we may use meta-facts **true(a)**, **false(a)**. Here **true(a)** is true if **a** is true and it is false if **a** is false or unknown, and **false(a)** is true if **a** is false and it is false if **a** is true or unknown.

The only downward transformation we use in this paper is the *downward assumption making transformation*. This is based on the binary generic meta-predicate **assumption**. If **a** is an atom of the object level subtask that is concerned, the meta-fact **assumption(a, pos)** means that currently "**a** is true" is an assumption for this object level subtask. Therefore **a** can be used in the reasoning of this object level subtask. Similarly, **assumption(a, neg)** means that $\neg$**a** is an assumption. We will illustrate this by our examples in later sections.

A second part of the description of the global level concerns the control flow. Here we use (supervisor) knowledge in rule-format that express what subtask should be activated, given a specific state of the subtasks. A simple example of such a rule is:

if **termination(subtask_A, success)**
then **next_subtask(subtask_B) and next_transformation(α)**

By use of this format the choice of the next subtask can be made conditional, depending on the success or failure of earlier tasks. For a more detailed treatment of supervisor

knowledge, see [7]. In general it is hard to extract supervisor knowledge from verbal protocols (without retrospection). Often subtasks are simply carried out, without explaining why: reasoning at the global (supervisor) level is left implicit.

2.3 Relations to other work

Essentially our approach combines a linguistic analysis with an approach based on task models, for instance as advocated by Breuker and Wielinga in [1]. Using the taxonomy of knowledge engineering techniques as described by Shadbolt and Burton [6], our approach can be classified as follows:

- the expert uses a language that is familiar to him or her (non-contrived)
- it concerns the analysis of protocols (and not interviews) (protocol analysis)
- the protocols are verbal (verbal)
- the ongoing problem solving process is analysed without taking into account retrospective comments on it (on-line)

This shows a similarity to the work of Johnson et al. [4]. However, not all of these qualifications may be essential to our approach. For example, it is an open question whether more contrived expert activities could be designed and included. As far as the last qualification is concerned, it could be interesting to investigate off-line style protocols as well (i.e. protocols where an expert gives retrospective comments and justifications related to what he or she was doing during the problem solving). Since retrospective activities often improve and clarify the logical structure of an argument, we expect that our approach is suitable to handle these types of protocols.

In addition to the approaches based on task models, as mentioned above, we use logical description techniques to analyse protocols (both for the global task level and for the local primitive subtask level). In this respect our analysis includes aspects known from the theory of argumentation (also called informal logic). In the theory of argumentation (for instance, see [13]) one discusses aspects of reasoning processes such as:

- is a given reasoning step a sound step, based on some logical (deduction) principle
- what are the global underlying presuppositions of the argument
- which local assumptions are made, and what is their scope within the argument; e.g., when are they withdrawn (for instance using reductio ad impossibile)

In the literature on the theory of argumentation much stress is put on the descriptions of the argument at the local level; for descriptions of this level also more formal logical principles are used. How to generate and focus a reasoning pattern (issues of structuring and controlling the reasoning) is not treated well. As an example, handling dynamic assumptions (introducing and later on withdrawing them) is only treated in an informal manner. Therefore, in our view a straightforward use of the theory of argumentation to analyse verbal protocols, without involving a task model is not adequate.

Our approach subsumes ideas from the theory of argumentation as well as the use of task models. Moreover, in addition we use formal descriptions both at the local level and at the global level. We think the use of formal descriptions has two main advantages:

- the results of an analysis are well-defined, less vague and may easier be compared to each other

- if a system has to be designed and implemented in a later phase of the work, then formal analyses of tasks and primitive subtasks can be used to provide automated support for this process of designing and implementing the system.

In fact for this second phase of the development path we use a framework for design and specification of modular knowledge-based systems: DESIRE (framework for DEsign and Specification of Interacting REasoning modules). This DESIRE framework has a formal foundation and fits very well to the approach to protocol analysis as described here.

3 An example analysis

In this section we illustrate our approach in some more detail by means of an example. This example concerns hypothetical reasoning. Hypothetical reasoning proceeds by assuming a set of assumptions for a while, and deriving what facts would follow from it. The derived facts may be evaluated; based on this evaluation one may decide to reject some of the assumptions or to choose a new set of assumptions. As an example, if one assumption "c is true" was chosen, and the facts derived from this assumption contradict information that we obtained from a different source, we may decide to reject the assumption "c is true" and to assume the converse "c is false". For instance, this pattern of reasoning occurs in the proof method called "indirect proof". Notice that in such reasoning patterns two levels of reasoning are involved: *object level reasoning*, where conclusions are drawn from the assumptions, and *meta-level reasoning* where the assumptions themselves are proposed and evaluated (also see [14]).

As an example we consider the following simple example reasoning pattern; actually, this protocol was not acquired by observation, but by introspection on our own reasoning experiences. We think it is not too unrealistic:

- I see no clouds
- suppose it will be raining soon
- then in any case there should be clouds
- but there are no clouds
- therefore it will not be raining soon

The following implicit underlying knowledge is left unspoken in this protocol:

- if it will be raining soon then there should be clouds

By retrospection we design a model for this reasoning pattern consisting of the task decomposition and subtask interactions as shown in Fig 1.

3.1 Descriptions at the local level: the subtasks

A more detailed description of the primitive subtasks is given by:

subtask 0: *external world* (observations from other source)
output atoms: **clouds_visible**

subtask 1: *deriving consequences*
input atoms: **rain_soon**
output atoms: **clouds_visible**

knowledge:

rain_soon → clouds_visible

The reasoning in this subtask is done exhaustively and data-driven.

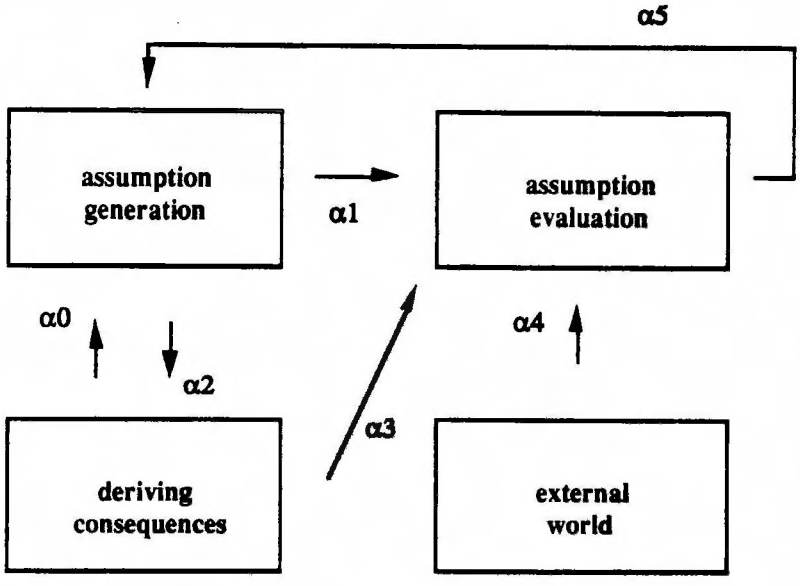


Fig 1 Task model for hypothetical reasoning

subtask 2: *assumption generation*

input atoms: **contradictory(rain_soon, pos), known(rain_soon)**

output atoms: **poss_assumption(rain_soon, pos), poss_assumption(rain_soon, neg)**

knowledge:

¬ known(rain_soon) → poss_assumption(rain_soon, pos)

contradictory(rain_soon, pos) → poss_assumption(rain_soon, neg)

The reasoning in this subtask will be done by data-driven reasoning, non-exhaustive.

Notice that we use the same name **rain_soon** both in the deriving consequences subtask (as a nullary relation name) and in the assumption generation subtask and the assumption evaluation subtask (as an object name). In the description of the transformations these names will be linked in the right manner. Similarly, we use the name **clouds_visible** differently in two different subtasks: in the deriving consequences subtask and in the assumption evaluation subtask.

subtask 3: *assumption evaluation*

input atoms: **assumed(rain_soon, pos), assumed(rain_soon, neg),**
derived(clouds_visible, pos),
observed(clouds_visible, pos), observed(clouds_visible, neg)

output atoms: **contradictory(rain_soon, pos), contradictory(rain_soon, neg)**

knowledge:

assumed(a, pos) ∧ derived(c, pos) ∧ observed(c, neg) → contradictory(a, pos)

The knowledge can be expressed in a generic form. Of course, other but similar generic rules can be added. For simplicity we do not list them.

The reasoning in this subtask will be done exhaustively, data-driven.

Notice that the knowledge as specified above is rather minimal: the example reasoning pattern can be described but not much more. We discuss the types of the transformations as depicted in Fig 1.

3.2 Description of the global level

First we give a description of the output-input connections between the subtasks.

0. Epistemic upward transformation α_0

   known(rain_soon) ⇔ known(rain_soon)

1. Object interaction α_1 (type O-O)

   poss_assumption(rain_soon, s) ⇔ assumed(rain_soon, s)

2. Downward assumption making transformation α_2

   poss_assumption(rain_soon, s) ⇔ assumption(rain_soon, s)

3. Epistemic upward transformation α_3

   true(clouds_visible) ⇔ derived(clouds_visible, pos)

4. Epistemic upward transformation α_4

   true(clouds_visible) ⇔ observed(clouds_visible, pos)

5. Object level transformation α_5

   contradictory(rain_soon, s) ⇔ contradictory(rain_soon, s)

A second part of the global description is that of the global dynamics of the reasoning pattern: a supervisor to control the activation of the subtasks. For instance, this supervisor should first activate the assumption generation subtask, next (if the first subtask terminated with success, as is the case in our mini-example) the deriving consequences subtask; if this succeeds (as is the case) then the other source subtask and the assumption evaluation subtask should be activated subsequently. Finally, activating the assumption generation subtask once again, in our mini-example provides the right assumption, namely that it will not rain soon. This can be reflected downwards. An example trace of such a reasoning pattern is depicted in Fig 2. Here between [..] the literals are given that currently hold in the information state of the subtask. The use of supervisor knowledge and the effects of reflective transformations have only been made visible as far as the consequences for the information states are concerned.

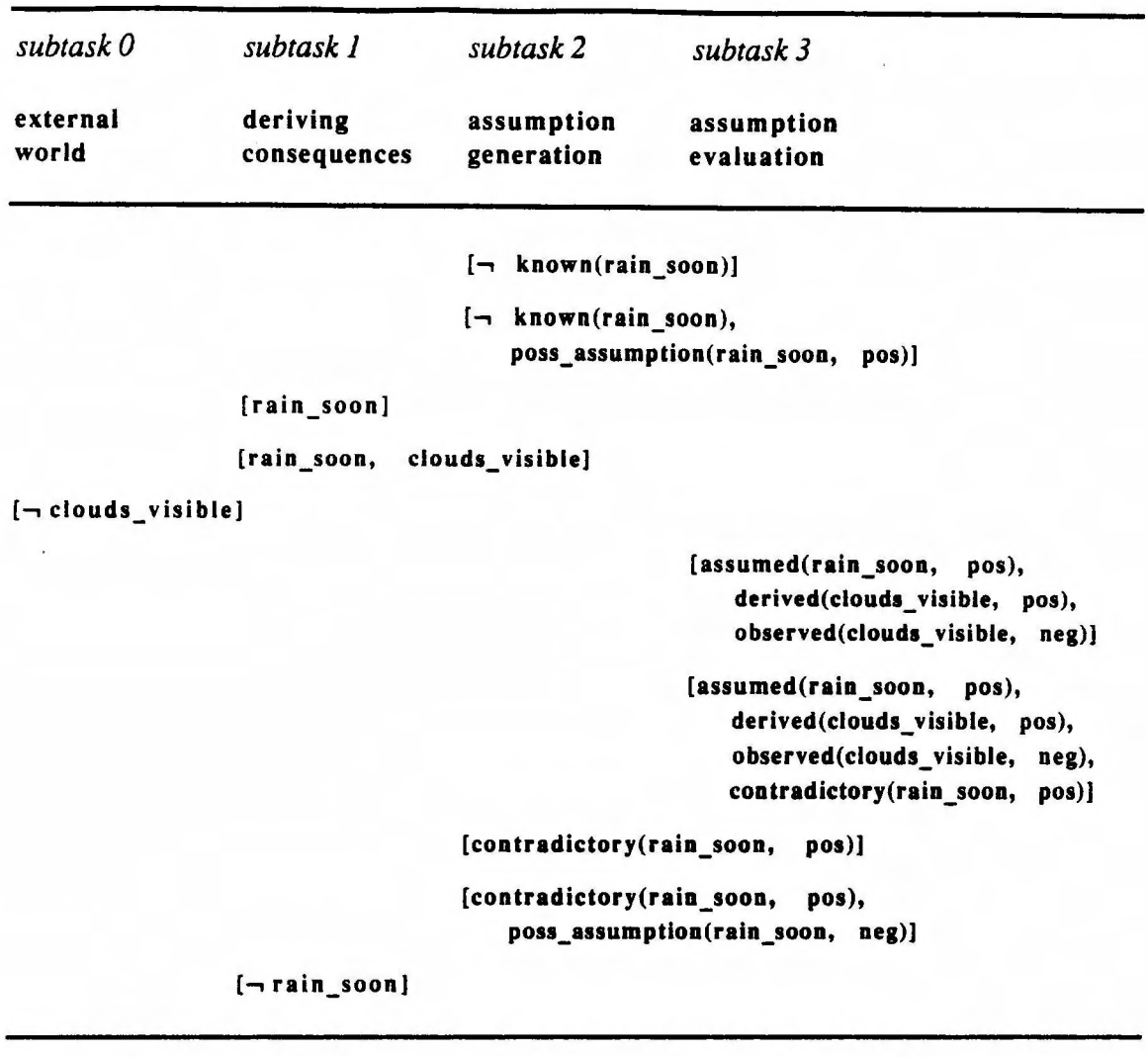


Fig 2 Trace of an example pattern of hypothetical reasoning

4 A more complex example

As a more complex example we analyse a simple version of the wise men's puzzle. In section 5 some empirical work on this puzzle is presented. We have chosen this example since it involves reflective reasoning patterns that cannot be globally expressed by standard logic.

The puzzle runs as follows. Each of the two wise men (A and B) has a hat that he cannot see himself. The only thing that is known beforehand is:

- the colour of each hat is white or black
- there is at least one white hat
- each wise man knows that the other person is able to reason logically
- each wise man can only observe the colour of the other wise man's hat

In the example situation analysed in this section both men have a white hat, without knowing this beforehand. First A is asked whether he knows what the colour of his hat

is. The answer of A is that he does not know. Then B is asked the same question. The answer of B is that he knows that his hat is white.

By introspection on our own reasoning experiences we have designed a model for the reasoning that is needed for B to draw his conclusion. Since the reasoning involves hypothetical reasoning, the task model described in section 3 is very useful as one of the building blocks. The task model for this case is shown in Fig 3.

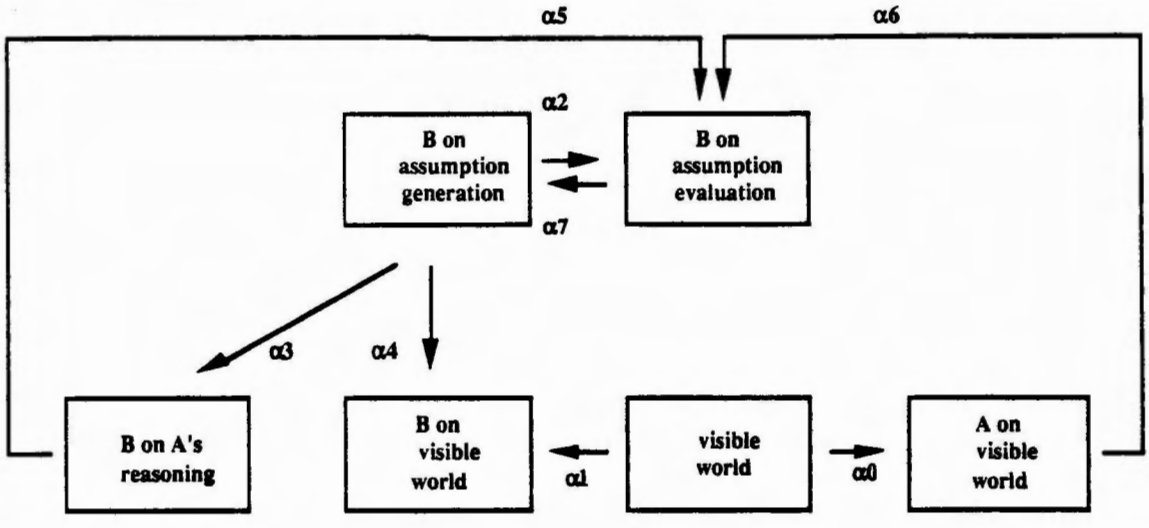


Fig 3 Task model for the wise men's puzzle

4.1 Description of the local level

Below we will give a short description of each of the subtasks.

subtask 0: *visible world*
output atoms: **white_A, black_A, white_B, black_B**
This subtask models the visible real world as can be seen.

subtask 1: *viewpoint A on the visible world*
input atoms: **white_B, black_B**
output atoms: **white_A, black_A**
knowledge : **black_B → white_A**
Using this knowledge A can reason about the visible world.

subtask 2: *viewpoint B on the visible world*
input atoms: **white_A, black_A**
output atoms: **white_B, black_B**
knowledge : **black_A → white_B**
Using this knowledge B can reason about the visible world.

Notice that both A and B can see a part of the visible world, but these parts differ. This difference is expressed by the different domains of the transformations from the external world to each of the two viewpoints. The following subtask models the reasoning of B about A's reasoning.

subtask 3: *viewpoint B on A*
input atoms: **A_observed(white_B), A_observed(black_B)**
output atoms: **A_derivable(white_A), A_derivable(black_A)**
knowledge :
The knowledge consists of the knowledge of A; we assume that B knows what knowledge A has. This is specified by **A_rule(X, Y)** denoting that A has the knowledge that **X** implies **Y**. Moreover, B has knowledge on how A reasons with his knowledge by chaining:

A_rule(black_B, white_A)
A_observed(X) → **A_derivable(X)**
A_derivable(X) ∧ **A_rule(X,Y)** → **A_derivable(Y)**

The fact **A_derivable(X)** denotes that A is able to derive statement **X**. The fact **A_observed(X)** states that fact **X** is observed by A.
The reasoning is exhaustive and data-driven.

subtask 4: *viewpoint B on assumption generation*
input atoms: **observed(A_says_white_A), contradictory(A_sees_black_B, pos)**
output atoms: **poss_assumption(A_sees_black_B, pos),**
poss_assumption(A_sees_white_B, pos),
best_assumption(A_sees_black_B, pos),
best_assumption(A_sees_white_B, pos)
knowledge :
observed(A_says_white_A) → **poss_assumption(A_sees_white_B, pos)**
¬ observed(A_says_white_A) → **poss_assumption(A_sees_black_B, pos)**
contradictory(A_sees_black_B, pos) → **best_assumption(A_sees_white_B, pos)**
contradictory(A_sees_white_B, pos) → **best_assumption(A_sees_black_B, pos)**
The reasoning is non-exhaustive, data-driven until a conclusion **best_assumption(X,pos)** is derived for some **X**.

subtask 5: *viewpoint B on assumption evaluation*
input atoms: **assumed(A_sees_white_B, pos), assumed(A_sees_black_B, pos),**
derived(A_says_white_A, pos),
observed(A_says_white_A, pos), observed(A_says_white_A, neg)
observed(A_says_black_A, pos), observed(A_says_black_A, neg)

output atoms: **contradictory(white_B, pos), contradictory(white_B, neg)**
knowledge :
The knowledge can be expressed in a generic form; it consists of a number of rules dealing with different types of conflicts; for simplicity we only mention:

assumed(a, pos) ∧ derived(c, pos) ∧ observed(c, neg) → contradictory(a, pos)
assumed(a, pos) ∧ ¬ derived(c, pos) ∧ observed(c, pos) → contradictory(a, pos)
The reasoning is exhaustively data-driven.

4.2 Description of the global level

We will discuss the interactions.

0. Object interaction α_0

white_B ⇔ white_B
black_B ⇔ black_B

1. Object interaction α_1

white_A ⇔ white_A
black_A ⇔ black_A

2. Object interaction α_2

poss_assumption(A_sees_white_B,pos) ⇔ assumed(A_sees_white_B,pos)
poss_assumption(A_sees_black_B,pos) ⇔ assumed(A_sees_black_B,pos)

3. Downward assumption making interaction α_3

poss_assumption(A_sees_white_B,pos) ⇔ assumption(A_observed(white_B), pos)
poss_assumption(A_sees_black_B,pos) ⇔ assumption(A_observed(black_B), pos)

4. Downward assumption making interaction α_4

best_assumption(A_sees_white_B, pos) ⇔ assumption(white_B, pos)
best_assumption(A_sees_black_B, pos) ⇔ assumption(black_B, pos)

5. Epistemic upward transformation α_5

true(A_derivable(white_B)) ⇔ derived(A_says_white_A, pos)
true(A_derivable(black_B)) ⇔ derived(A_says_black_A, pos)

6. Epistemic upward transformation α_6

true(white_A) ⇔ observed(A_says_white_A, pos)
not true(white_A) ⇔ observed(A_says_white_A, neg)
true(black_A) ⇔ observed(A_says_black_A, pos)
not true(black_A) ⇔ observed(A_says_black_A, neg)

7. Object level transformation α_7

contradictory(white_B, pos) ⇔ contradictory(white_B, pos)
contradictory(white_B, neg) ⇔ contradictory(white_B, neg)

As in section 3, it is also necessary to specify a supervisor to control the activation of the subtasks. For instance, the supervisor should first activate the viewpoint A subtask, that requests its part of the visible world and then is not able to derive anything. Next the viewpoint B on the visible world can be activated; also this subtask is not able to derive anything. Next the viewpoint B on assumption generation subtask may be activated, deriving that **A_sees_black_B** is a possible assumption. Next the viewpoint B on A subtask can be activated, deriving that **A_derivable(white_A)**. If this succeeds (as is the case in our mini-example) then the viewpoint B on assumption evaluation subtask should be activated. This leads to a conclusion that **A_sees_black_B** is contradictory. Finally, activating the viewpoint B on assumption generation subtask once again, in our mini-example provides the right assumption, namely that **A_sees_white_B**. This conclusion is reflected downwards. In Fig 4 this reasoning trace is depicted. To maintain a convenient layout, in this figure we abbreviated the atom names. For example, instead of **observed(A_says_white_A)** we will use **obs(A_says_wh_A)** .

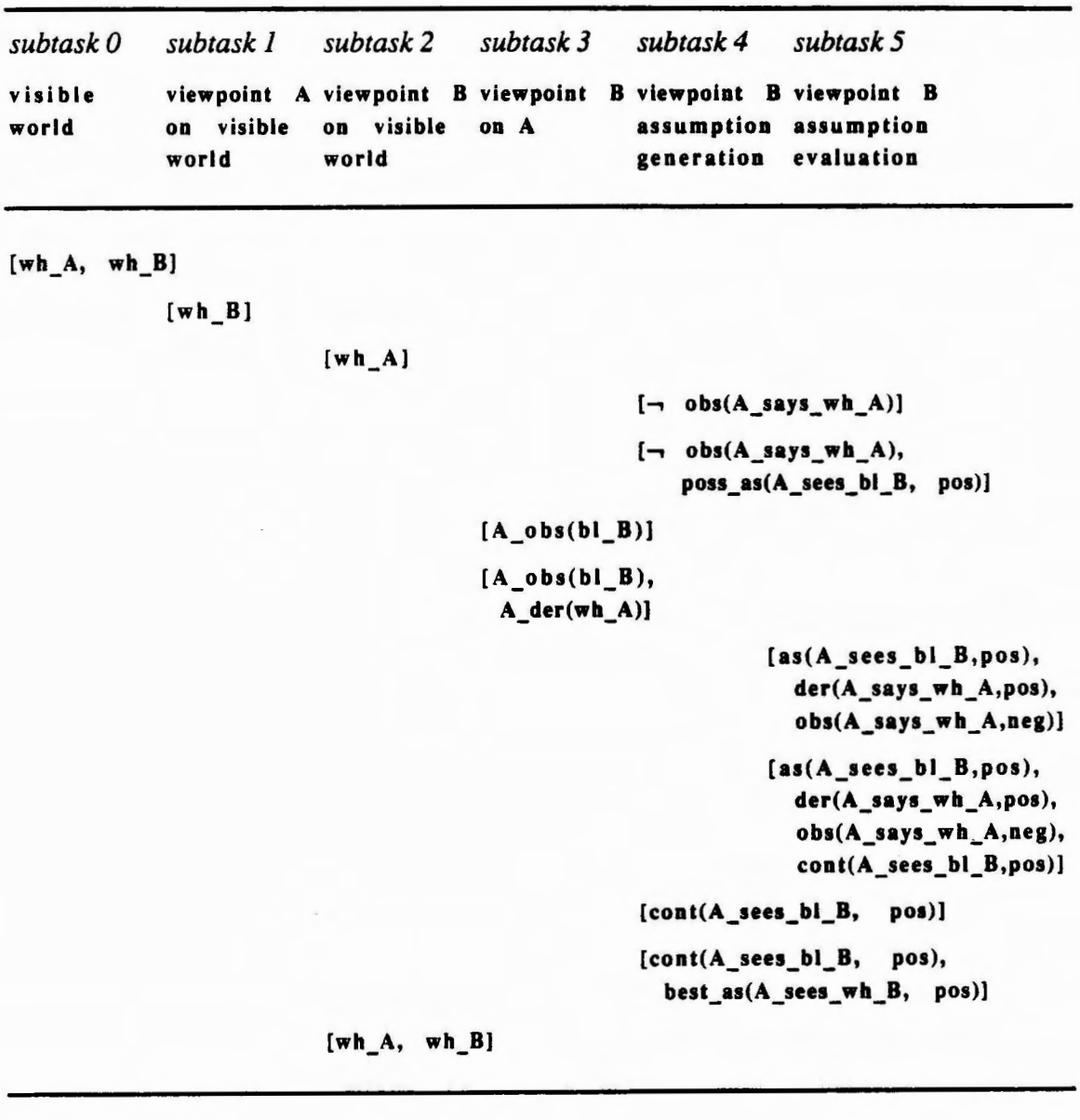


Fig 4 Reasoning trace for the wise men's puzzle

It is not difficult to describe the reasoning traces in case of one white hat and one black one. In the case that A has a black hat, B's reasoning about the visible world subtask will draw the conclusion that B's hat is white. If B has a black hat, then A immediately derives that A has a white hat. On the basis of this information B generates the assumption that A sees a white hat of B. This assumption is tried in viewpoint B on A's reasoning, not leading to any derived fact. Therefore B generates the assumption that B has a black hat in this case.

The representation of the knowledge involved in this wise men's example also has been described by Attardi and Simi in their knowledge representation language OMEGA (see [15]). Other related references on reasoning across viewpoints and contexts are [16] and [17].

5 Empirical work

The task model for the wise men's puzzle, as described in the previous section, has been designed on the basis of introspection and retrospection. It seems plausible, but it is not clear whether it conforms to the reasoning pattern exhibited by other human reasoners. To verify the psychological validity of our task model we asked a subject to solve the wise men's puzzle and to think aloud while doing so. The thinking aloud protocol acquired from this experiment has been coded in terms of our task model and analysed to see to what extent the subject's reasoning pattern matches the model's.

Our subject was given the following description:

> This puzzle is about two wise men, A and B, each of which is wearing a hat. Each hat is either black or white but at least one of the hats is white. Each wise man can *only* observe the colour of the *other* wise man's hat. Both wise men are able to reason logically and they know this from each other.

The subject was asked to solve four variants of the puzzle. For each variant he was given the colour of A's hat and whether A knows this colour. Note that one of the combinations of A's hat and A's knowledge on his own hat is actually impossible in the given context. It is left as a puzzle for the reader to find out which of the variants is impossible.

For each variant of the puzzle the subject was asked to reason as if he was B and to reason the colour of B's hat given the colour of A's hat and A's knowledge about his own hat. The subject was given instructions to think aloud and the protocol was recorded on a tape recorder.

In the following we present a number of fragments of the obtained protocols.

situation 1:

A is wearing a black hat
A does not know that he is wearing a black hat

protocol fragment 1:

19. If A is wearing a black hat
20. and B sees this
21. then B knows that his hat has to be white
22. because there must be at least one white
23. and then that is the answer.

situation 2:

A is wearing a white hat
A does not know that he is wearing a white hat

protocol fragment 2:

1. A sees either a white hat or a black hat of B.
2. If he sees a black hat of B
3. then he knows that he wears a white one
4. and then he also knows what colour he wears
5. that is the white one,
6. so in that case he doesn't answer "I don't know"
7. so A must see a white hat.
8. then he doesn't know
9. since there can be two white hats involved
10. it can be also the case that A wears a black hat
11. and B a white hat
12. so A doesn't know what hat he is wearing
13. and that means that A, as I mentioned before, must have seen a white hat,
14. so B can conclude, after A's answer that he is wearing a white hat.

situation 3:

A is wearing a white hat
A knows that he is wearing a white hat

protocol fragment 3:

2. If A knows the colour of the hat he is wearing
3. then he must have seen a black one,
4. because if B wears a white one
5. then there can be another white one involved
6. or there can be a black one involved
7. so you can exclude this possibility
8. we may assume that B is wearing a black hat
9. and that A concludes from this "I am wearing a white hat".

situation 4:

A is wearing a black hat
A knows that he is wearing a black hat

protocol fragment 4:

3. if A, A says, "I know the colour of my hat
4. and it is black "

5. if A sees a black hat
6. then he doesn't know which hat he is wearing
7. yes, he does know
8. then he is wearing the white one

9. if B is wearing a white hat
10. then A can wear either a white hat or a black hat
11. so, in my opinion, A can't claim he is wearing a black hat

For the analysis of the protocols we will code or interprete each fragment in terms our task model. Each line of a fragment will be labeled with the element(s) of the task model it corresponds to. The following notation is used for these encodings:

subtask: *<subtask name>*	<lines in protocol>
input	
< information type 1>	<lines in protocol>
(<true input literals of this type for the subtask>)	
:	
< information type n>	<lines in protocol>
(<true input literals of this type for the subtask>)	
output	
< information type >	<lines in protocol>
(<true output literals of the subtask>)	

This results in the following encodings of the above protocol fragments.

code fragment 1:

subtask: *B on visible world*	19-23
input	
B's observations	19-20
(**black_A**)	
output	
B's conclusions on world	21
(**white_B**)	

code fragment 2:

subtask: *B on assumption generation*	1-2

input

output
B's assumption on A's observations 2
(poss_assumption(A_sees_black_B, pos))

subtask: *B on A's reasoning* 2-5
input
B's assumption on A's observations 2
(A_observed(black_B))
output
B's hypothetical conclusions on A's reasoning 3-5
(A_derivable(white_A))

subtask: *B on assumption evaluation* 2-6, 12
input
B's hypothetical conclusions on A's reasoning 3-5
(derived(A_says_white_A, pos))
B's observational information on A's reasoning 12
(observed(A_says_white_A, neg))
B's assumption on A's observations 2
(assumed(A_sees_black_B, pos))
output
contradiction of B's assumption 2, 6
(contradictory(A_sees_black_B, pos))

subtask: *B on assumption generation* 2, 6-7, 13
input
contradiction of B's assumption 2, 6
(contradictory(A_sees_black_B, pos))
output
B's best assumption on A's observations 7, 13
(best_assumption(A_sees_white_B, pos))

subtask: *B on A's reasoning* 7-11
input
B's assumption on A's observations 7
(A_observed(white_B))
output
B's hypothetical conclusions on A's reasoning 8
( )

subtask: *B on assumption evaluation* 7-8, 12
input
B's hypothetical conclusions on A's reasoning 8
(not derived(A_says_white_A, pos))
B's observational information on A's reasoning 12
(observed(A_says_white_A, neg))
B's assumption on A's observations 7
(assumed(A_sees_white_B, pos))
output
contradiction of B's assumption 7, 13
( )
subtask: *B on visible world* 7, 13-14
input
B's best assumption on A's observations 7,13
(assumed(white_B, pos))

output
B's conclusions on world 14
(white_B)

code fragment 3:

subtask: *B on assumption generation* 4
input

output
B's assumption on A's observations 4
(poss_assumption(A_sees_white_B, pos))

subtask: *B on A's reasoning* 4-6
input
B's assumption on A's observations 4
(A_observed(white_B))
output
B's hypothetical conclusions on A's reasoning (5-6)
( )

subtask: *B on assumption evaluation* 2, 4, 7
input
B's hypothetical conclusions on A's reasoning (5-6)
(not derived(A_says_white_A, pos)),
B's observational information on A's reasoning 2
(observed(A_says_white_A, pos))
B's assumption on A's observations 4
(assumed(A_sees_white_B, pos))
output
contradiction of B's assumption 4, 7
(contradictory(A_sees_white_B, pos))

subtask: *B on assumption generation* 4, 7-8
input
contradiction of B's assumption 4, 7
(contradictory(A_sees_white_B, pos))
output
B's best assumption on A's observations 8
(best_assumption(A_sees_black_B, pos))

subtask: *B on A's reasoning* 8, 9
input
B's assumption on A's observations 8
(A_observed(black_B))
output
B's hypothetical conclusions on A's reasoning 9
(A_derived(white_A))

subtask: *B on assumption evaluation* 2, 8-9
input
B's hypothetical conclusions on A's reasoning 9
(derived(A_says_white_A, pos))
B's observational information on A's reasoning 2
(observed(A_says_white_A, pos))
B's assumption on A's observations 8
(assumed(A_sees_black_B, pos))

output
contradiction of B's assumption 8-9
( )
subtask: *B on visible world* 3, 8
input
B's best assumption on A's observations 8
(assumed(black_B, pos))
output
B's conclusions on world 3
(black_B)

code fragment 4:

subtask: *B on assumption generation* 5
input

output
B's assumption on A's observations 5
(poss_assumption(A_sees_black_B, pos))

subtask: *B on A's reasoning* 5-8
input
B's assumption on A's observations 5
(A_observed(black_B))
output
B's hypothetical conclusions on A's reasoning 7-8
(A_derived(white_A))

subtask: *B on assumption evaluation* 3-8
input
B's hypothetical conclusions on A's reasoning 7-8
(derived(A_says_white_A, pos))
B's observational information on A's reasoning 3, 4
(observed(A_says_white_A, neg))
B's assumption on A's observations 5
(assumed(A_sees_black_B, pos))
output
contradiction of B's assumption 3-4, 8
(contradictory(A_sees_black_B, pos))

subtask: *B on assumption generation* 3-4, 8-9
input
contradiction of B's assumption 3-4, 8
(contradictory(A_sees_black_B, pos))
output
B's best assumption on A's observations 9
(best_assumption(A_sees_white_B, pos))

subtask: *B on A's reasoning* 9, 10
input
B's assumption on A's observations 9
(A_observed(white_B))
output
B's hypothetical conclusions on A's reasoning 10
( )

subtask: *B on assumption evaluation*	3-4, 9-10
input	
B's hypothetical conclusions on A's reasoning	10
(not derived(A_says_white_A, pos))	
B's observational information on A's reasoning	3-4
(observed(A_says_black_A, pos))	
B's assumption on A's observations	9
(assumed(A_sees_white_B, pos))	
output	
contradiction of B's assumption	10-11
(contradictory(A_sees_white_B, pos))	

From these encodings it can be concluded that the subject's decomposition of the complex reasoning task into subtasks matches the decomposition that underlies the task model we designed on the basis of our own reasoning experiences. However, it also appeared that the order in which our subject tackled the subtasks differs somewhat from the order specified by our task model. In other words, the control structure of our task model does not completely match the reasoning pattern of our subject.

The main difference is that the subject is more careful in drawing a final conclusion than the model. When in the task model an assumption for the colour of B's hat leads to a contradiction, it is concluded that B wears a hat of the inverse colour. Our subject appeared to be more detained. After deriving a contradiction from an assumption for B's hat he did *not* always conclude blindly that the colour of B's hat must be the inverse of his assumption, but he checked first whether this inverse colour would not lead to a contradiction. This behaviour is of course justified by the possibility of an impossible situation.

A consequence of this careful reasoning is that the subject did not base his initial assumption on knowledge about A, as specified in the task model, but made a rather arbitrary initial assumption, knowing that he would have to check the other possibility anyhow. Therefore in situation 4 the subject gave the correct solution whereas our task model would have failed to do so.

All in all it appeared that our task model is psychologically plausible to a large extent. The difference between subject and model can be explained by the fact that the model was not designed to reason about situations that cannot occur.

6 Conclusions

The examples that have been analysed above show us that interesting complex reasoning patterns can be modelled by interacting reasoning modules according to our logical approach. Compared to the approach where a reasoning pattern is described within one logic, this gives an alternative (modular) approach to the modelling of reasoning patterns. This modular approach enables one to describe the control aspects of the reasoning in an easy manner. Since in complex reasoning patterns (for instance involving reflective aspects) often procedural and control aspects play an important role, it is an important advantage of our approach that these can be described. Expressed in logical terms, we shift our attention from studying the properties of one logic to studying the properties of

relations between a number of logics. These relations can have both interesting declarative aspects and procedural aspects and combinations of these (i.e., in reflective relations between logics) and are worth being investigated further.

Acknowledgements

We thank Arnold Wentholt for reading an earlier draft of this document and having some discussions about the material that is included.

References

[1] Breuker, J.A., Wielinga, B.J. (1985). Interpretation of verbal data for knowledge acquisition. In: O'Shea T. (ed) *Advances in Artificial Intelligence*, Elsevier Science Publishers, pp. 3-12.

[2] Edin, G., Rooke, P., Breuker, J., Rangecroft, T., Hayes F. (1987). *The KADS handbook*. Esprit Project P1098, Deliverable J1-PartIII Ch 1,2, Sciton Ltd.

[3] Ericsson, K.A., Simon, H.A. (1984). *Protocol analysis*, M.I.T. Press.

[4] Johnson, P.E., Zualkerman, I., Garber, S. (1987). Specification of Expertise, *International Journal of Man-machine studies*, 26 , pp. 161-181.

[5] Kuipers, B., Kassirer, J.P. (1987). Knowledge acquisition by analysis of verbatim protocols. In: Kidd, A.L. (ed) *Knowledge Acquisition for Expert Systems: a Practical Handbook*, Plenum Press, pp. 45-71.

[6] Shadbolt, N., Burton, A.M. (1989). Knowledge elicitation. In: Wilson, J., Corlett, N.(eds). *Evaluation of Human Work: Practical Ergonomics Methodology*. Taylor and Francis.

[7] Kowalczyk, W., Treur, J. (1990). On the use of a formalized generic task model in knowledge acquisition. In: Wielinga, B., Boose, J., Gaines, B., Schreiber, G., Someren, M. van (eds) *Proc. 4th European Knowledge Acquisition Workshop*, EKAW-90, IOS Press, pp.198-221.

[8] Treur, J. (1990). Declarative functionality descriptions of interactive reasoning modules, Report IR-237, Vrije Universiteit Amsterdam, Department of Mathematics and Computer Science.

[9] Langevelde, I.A. van (1991). On the link between chaining and resolution; a possible justification for the rule format, Report, Vrije Universiteit Amsterdam, Department of Mathematics and Computer Science.

[10] Treur, J. (1989). A logical analysis of design tasks for expert systems, *Int. Journal of Expert Systems*, 2 , pp. 233-253.

[11] Treur, J. (1991). On the use of reflection principles in modelling complex reasoning, *Int. Journal of Intelligent Systems*, 6, pp. ??-??.

[12] Treur, J. (1990). Interaction types and chemistry of generic task models, Report IR-235, Vrije Universiteit Amsterdam, Department of Mathematics and Computer Science.

[13] Fogelin, R. (1982). *Understanding Arguments: an introduction to informal logic*, 2nd edition, Harcourt Brace Jovanovich.

[14] Tan, Y.H., Treur, J. (1989). A logical description of a bimodular system for nonmonotonic reasoning with dynamic assumptions, Report PE8907, University of Amsterdam, Programming Research Group.

[15] Attardi, G., Simi, M. (1984). Metalanguage and reasoning across viewpoints, In: *Proc. European Conf. on AI*, ECAI-84.

[16] Bowen, K.A. (1985). Meta-level programming and knowledge representation, *New Generation Computing*, 3, pp. 359-383.

[17] Giunchiglia, F., Weyhrauch R.W. (1988). A multi-context axiomatization of inessential non-monotonicity. In: Maes, P., Nardi, D. (eds) *Meta-level artchitectures and reflection*, Elsevier Science Publishers, pp. 271-285.

Eliciting constraints and non-monotonic preferences for multi-criterion decision making

David McSherry

Computing Department
Lancaster University
Lancaster
LA1 4YR

Abstract

The elicitation of user priorities and preferences, though largely a neglected issue in KBS research, is essential to provide effective knowledge-based support for multi-criterion decision making. For applications involving large numbers of alternatives, the elicitation of absolute constraints may also be necessary to reduce the number of alternatives to a manageable size by the elimination of those which fail to meet the requirements of the user. This paper describes techniques for the elicitation of user priorities and preferences, including non-monotonic preferences, and their incorporation in the reasoning process. Techniques for the elicitation of absolute constraints are also described. The techniques have been implemented in an integrated knowledge elicitation and consultation environment called McExpert.

1 Introduction

While the explicit representation of domain knowledge is considered an essential characteristic of knowledge-based systems (KBS), relatively little effort has been made to explicitly represent the preferences [1] or priorities [3,4] of the user. The neglect of these issues seriously limits the effectiveness of KBS for applications involving multi-criterion decision making (MCDM), including countless problems in business, commerce and industry as well as many of the important decisions which people make in their daily lives, such as choosing a career or buying a house.

Typically in MCDM, a large number of alternatives must be evaluated before an informed decision can be made. Before the alternatives can be meaningfully compared, the decision maker must identify his or her priorities (i.e. the factors which will influence the decision and their relative importance). The decision maker must also identify his or her preferences with respect to each decision criterion. Preferences which involve subjective judgement can often be expressed only in terms of a relative rating of the alternatives. For many decisions facing the consumer, however, relevant data may be available from consumer magazines or other sources, thus enabling preferences to be expressed in terms of the preferred values of attributes.

Existing KBS techniques, although applicable in certain stages of the MCDM process (such as the application of domain expertise in the evaluation of alternatives), are of limited usefulness for selecting the alternative which most closely matches the requirements of the decision maker. Typically they rely on heuristics which make implicit assumptions about the user's priorities and preferences in order to avoid the combinatorial explosion which results from the expression of multiple decision criteria in rule-based formalisms [3].

The Analytical Hierarchy Process (AHP) provides a method of weighting decision alternatives according to the priorities and preferences of the decision maker to produce an overall rating of the alternatives [5]. Factors which influence the decision and subsidiary factors are represented in the form of a decision hierarchy. The decision maker's priorities are established by pairwise comparison of factors. A matrix of priorities is constructed of which each element (a number between 1 and 9) represents the relative importance of the corresponding row and column factors. Similarly, the preferences of the decision maker are established by pairwise comparison of alternatives with respect to each decision criterion.

For many applications involving MCDM, the AHP has its own limitations. Pairwise comparison of alternatives by the decision maker is unrealistic when the number of alternatives is large, unnecessary when relevant data are available (e.g. from consumer magazines), and impossible when professional expertise is needed to evaluate alternatives. Provided the decision maker may be assumed consistent in his or her allocation of priorities, moreover, much of the information contained in the priority matrices of the AHP is redundant. In these circumstances, their construction imposes an unnecessary burden on the decision maker.

McSherry [3] describes how the limitations of the AHP can be overcome by the use of KBS techniques to enable domain knowledge to be incorporated in the reasoning process and by the development of more efficient techniques for the elicitation of user priorities and preferences. Modified AHP techniques have been developed which exploit the redundancy inherent in the construction of priority matrices. The priorities of factors,

for example, are elicited much more efficiently by comparison with the factor considered most important by the decision maker than by exhaustive pairwise comparison. The techniques have been implemented in an integrated knowledge elicitation and consultation environment for MCDM called McExpert. The present paper describes several enhancements of the tools and techniques provided in McExpert to support the MCDM process, including techniques for the elicitation of factual information, absolute constraints, and non-monotonic preferences.

Section 2 describes how a decision hierarchy is constructed in McExpert for the problem of choosing a second-hand car, an application which provides a source of examples throughout the paper. Techniques for eliciting factual information are described in Section 3. Section 4 describes techniques for the elicitation of absolute constraints and their incorporation in the reasoning process. Section 5 describes the tools provided in McExpert for the elicitation of priorities and preferences and how they have been extended to enable preferences to be elicited indirectly in terms of the preferred values of attributes. Techniques for the elicitation of non-monotonic preferences are described in Section 6. Section 7 describes the McExpert consultation environment and the facilities it provides for sensitivity analysis.

2 Building a decision hierarchy

Written in LPA MacPROLOG for the Apple Macintosh, McExpert (Multi-criterion Expert) provides an integrated knowledge elicitation and consultation environment for MCDM. A logic programming adaptation of the AHP is combined with a PROLOG meta-interpreter [3] to enable domain facts and rules to be incorporated in the reasoning process.

McExpert is designed to enable all the details of the problem (objective, factors, alternatives, constraints, priorities, preferences, facts and rules) to be elicited at consultation time. Usually, however, it is more practical for details which do not depend on the priorities and preferences of the user (such as domain facts and rules, alternatives, and possibly some of the decision criteria) to be elicited in advance from a domain expert.

The example domain of choosing a second-hand car is typical of countless consumer decisions which involve MCDM. Before an informed decision can be made, many factors, some of them involving subjective judgement, must be taken into consideration. A hierarchical representation of factors which influence the decision-making objective (Figure 1) is built dynamically as details of the problem domain are elicited. The choice of factors, though subjective, would typically include performance, type of car, reliability and price. Subsidiary factors influencing performance might include top speed, fuel

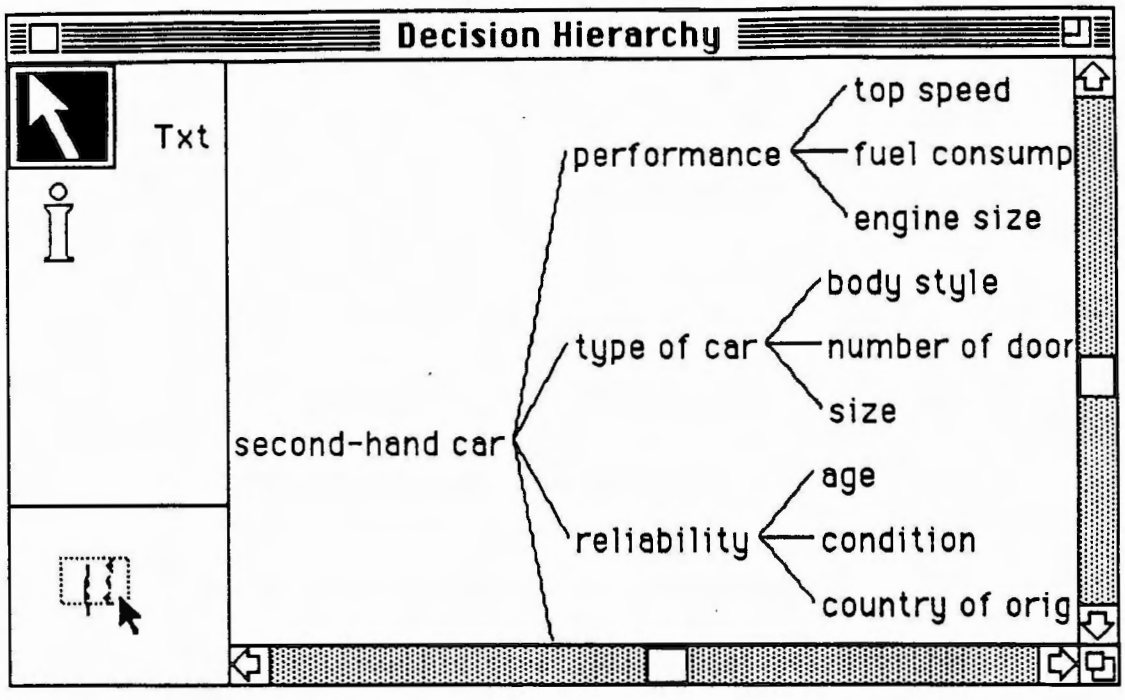


Figure 1. A decision hierarchy for choosing a second-hand car

consumption, and engine size. The leaves of the decision hierarchy are the basic decision criteria for the evaluation and comparison of alternatives.

As Figure 2 illustrates, building the decision hierarchy does not require interactive graphics skills. To extend the hierarchy, the user need only select an existing factor from a menu and then identify the subsidiary factors. Subsidiary factors which influence condition, for example, might include recorded mileage, mechanical condition and condition of the bodywork.

3 Eliciting factual information

In the example domain, the alternatives are the second-hand cars which are currently available, say from a local dealership (or advertised for private sale). Often they can be identified uniquely by the year of manufacture, make, and model of the car e.g.

1988 Ford Escort
1986 Peugeot 205

Relevant facts about the alternatives such as engine size, price, and country of origin can be stored in the knowledge base as facts such as:

engine size **of** 1988 Ford Escort **is** 1300
country of origin **of** 1988 Ford Escort **is** Great Britain

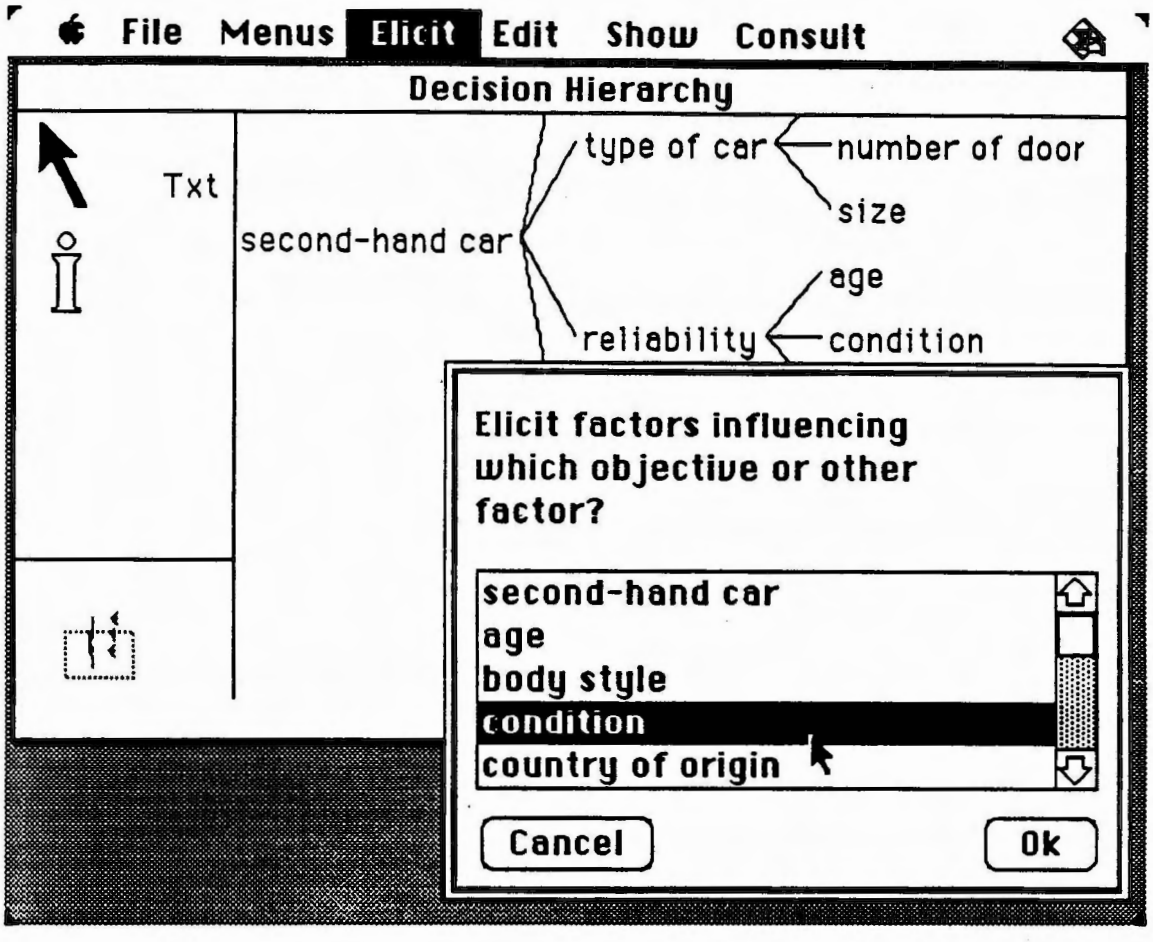


Figure 2. Extending the decision hierarchy

Figure 3 illustrates how such facts are elicited in McExpert. In the case of a non-numeric attribute (such as country of origin), the user can select from existing (previously elicited) values of the attribute.

4 Eliciting absolute constraints

For applications involving large numbers of alternatives, the elicitation of absolute constraints may be necessary to reduce the number of alternatives to a manageable size by the elimination of those which fail to meet the requirements of the decision maker. Two kinds of absolute constraints can be elicited in McExpert. For a numeric attribute such as price, the user can specify an upper limit, a lower limit, or a range of acceptable values (Figure 4). If values of the attribute are stored as facts in the knowledge base, the upper and lower limits are set by default to the highest and lowest values among all the alternatives.

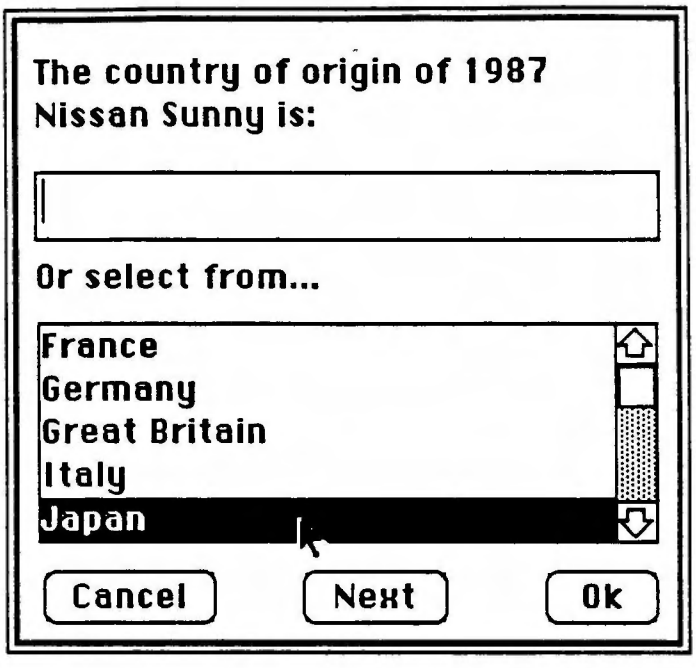


Figure 3. Eliciting factual information

In the case of a non-numeric attribute such as country of origin, the user can specify any subset of its values by selecting from a multiple-choice menu (Figure 5). Before confirming a constraint, the user can view the list of alternatives (Figure 6) which would satisfy the constraint.

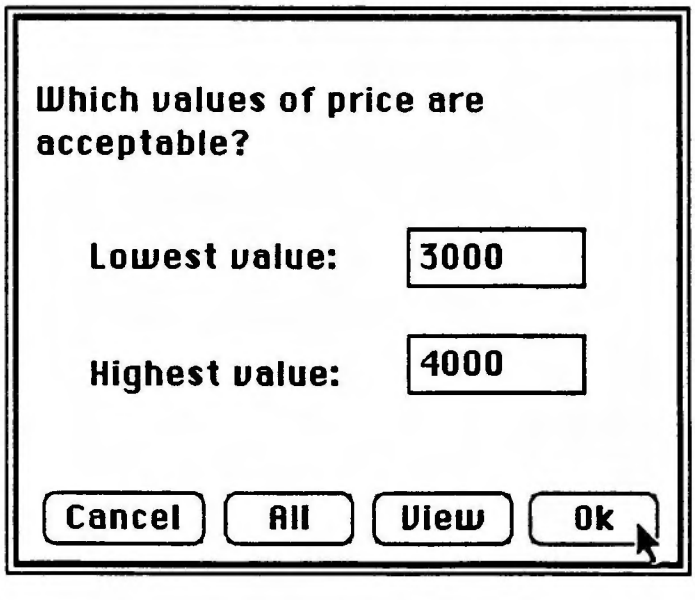


Figure 4. Eliciting numeric constraints

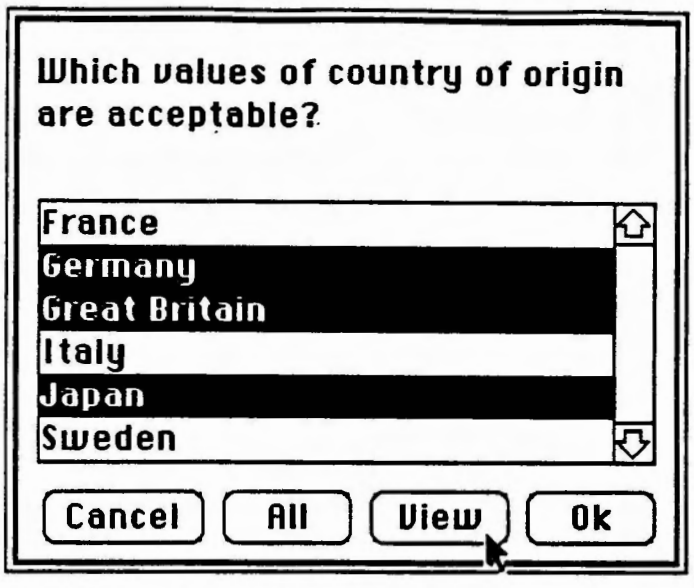


Figure 5. Eliciting non-numeric constraints

5 Eliciting priorities and preferences

Having identified the factors which influence the decision, the decision maker must identify his or her priorities (i.e. the relative importance of the factors). If subsidiary factors which influence another factor have been identified, then the decision maker must also decide the relative importance of the subsidiary factors. In McExpert, the priorities of factors are elicited by comparison with the factor considered most important by the decision maker. For example, if engine size is considered the most important factor influencing performance (Figure 7), it is allocated a priority of 1 and the priorities of each of the remaining factors is elicited as illustrated in Figure 8.

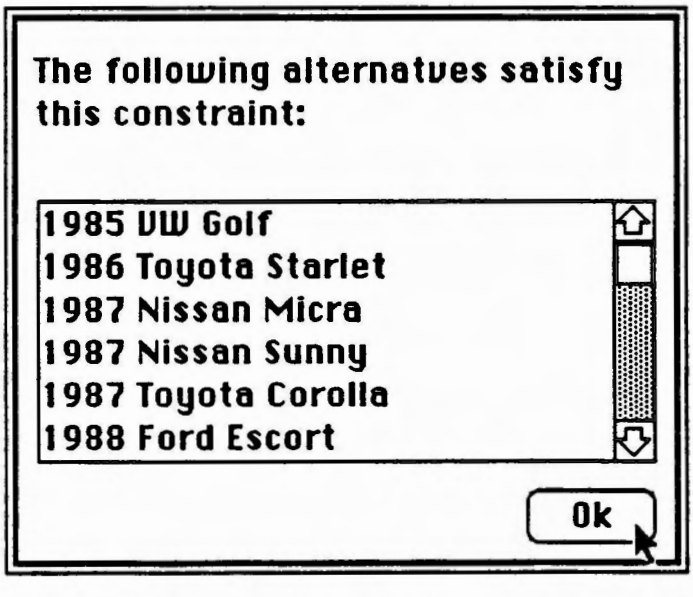


Figure 6. Viewing alternatives which satisfy a constraint

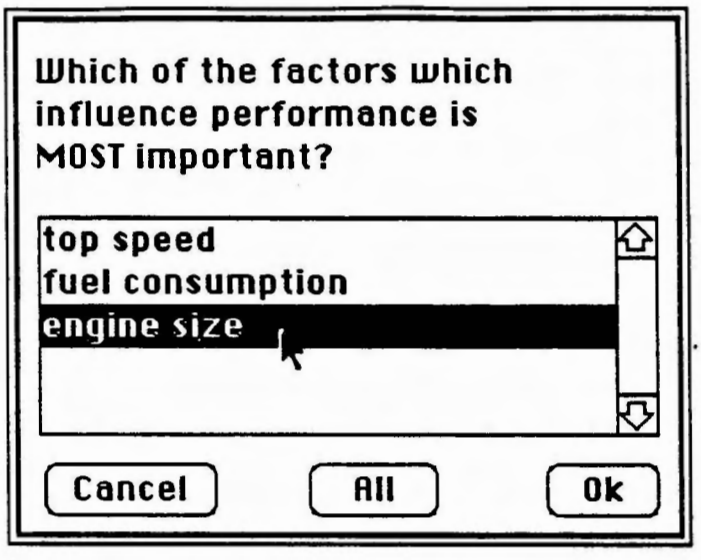


Figure 7. Eliciting the most important factor

In Figure 8, the options available to the decision maker correspond to the standard priority scores 1, 3, 5, 7 and 9 of the AHP [5]. By selecting two adjacent radio buttons, the decision maker can choose instead an intermediate score such as 2 or 4.

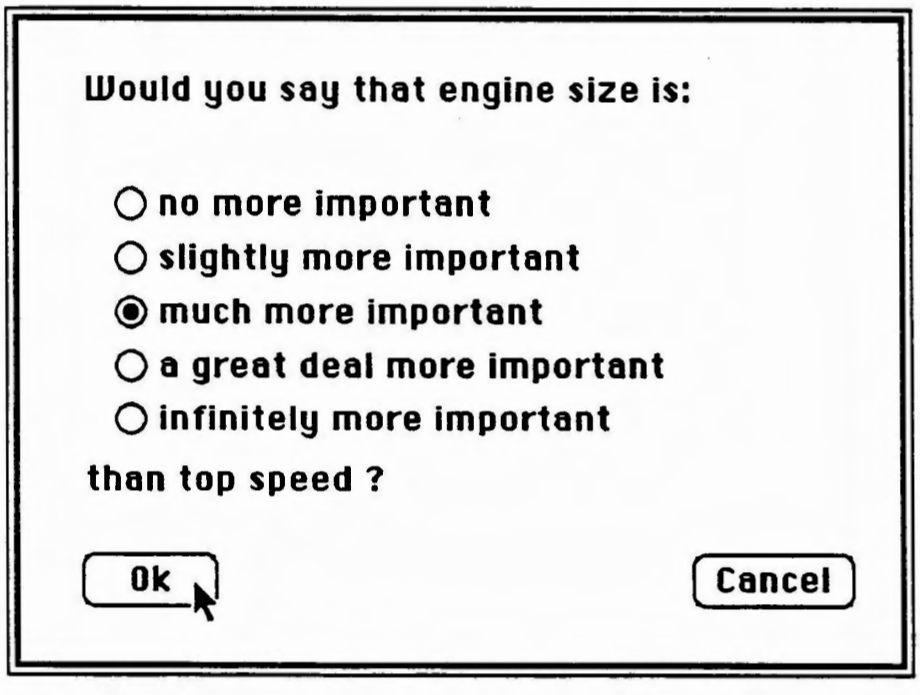


Figure 8. The priorities of factors are elicited by comparison with the most important factor.

Before the alternatives can be meaningfully compared, the decision maker must also identify his or her preferences. Provided the necessary information about alternatives is stored in the knowledge base, preferences can be expressed in terms of the preferred values of attributes. Techniques for the elicitation of preferences with respect to numeric attributes are described in the following section. For non-numeric attributes, McExpert combines a rapid approach to the elicitation of approximate preferences with the use of interactive graphics to increase the precision of the elicited preferences.

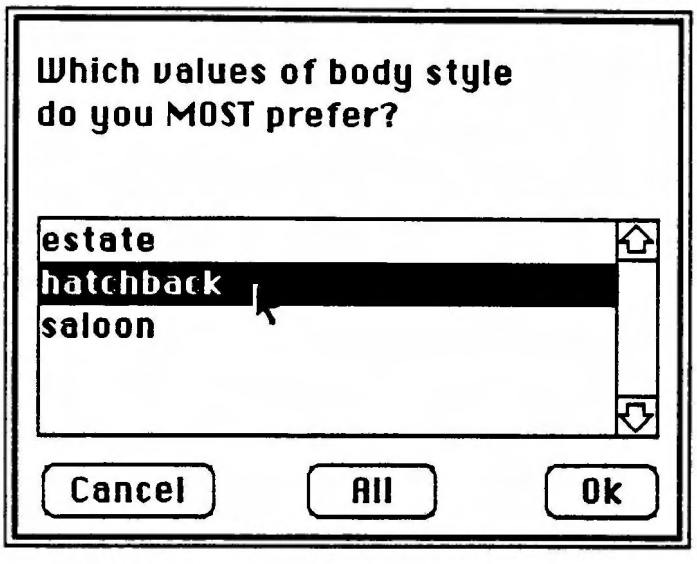


Figure 9. Eliciting preferences

By selecting from a multiple-choice menu (Figure 9) the decision-maker is invited to indicate the MOST preferred value of the attribute. By selecting more than one value, the user can indicate that the selected values are equally preferred. The selected values are assigned a preference score of 1 and removed from the menu. The user is then invited to indicate which of the remaining values he or she most prefers. The values selected on this occasion are assigned a preference score of 2 and these in turn are removed from the menu. This process continues until no further values remain (or just one remains).

The above technique enables approximate preferences to be elicited rapidly. A tool called the Preferences Editor (Figure 10) provides an interactive graphics interface for increasing the precision of elicited preferences. The preference scores correspond to the standard scores 1, 2, 3 . . . 9 of the AHP [5]. Thus *hatchback*, with a preference score of 1, is the most preferred body style. It is preferred twice as much as either of the alternative values *saloon* or *estate*. Using the tools provided in the tool pane at the left of the window, the user can increase or decrease the preference score of any value by clicking its button.

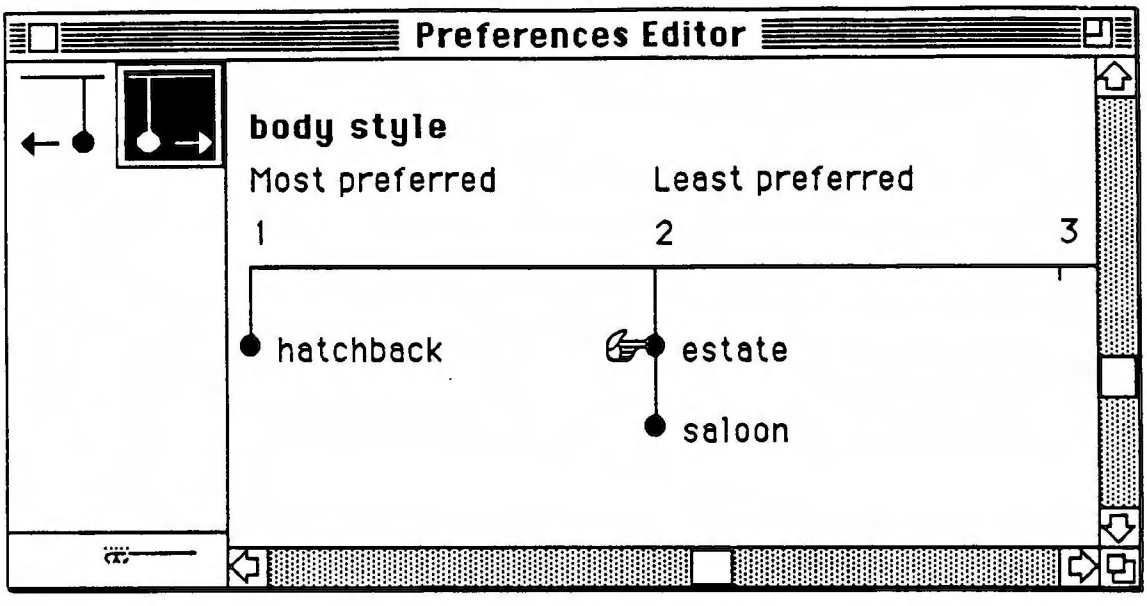


Figure 10. Increasing the precision of elicited preferences

In the example of Figure 10, the user has selected the tool (highlighted in the tool pane) for increasing preference scores and is about to increase the preference score of *estate*. The effect of this action will be to move *estate* and its button to the right by one place and increase its preference score to 3 (making it the least preferred body style).

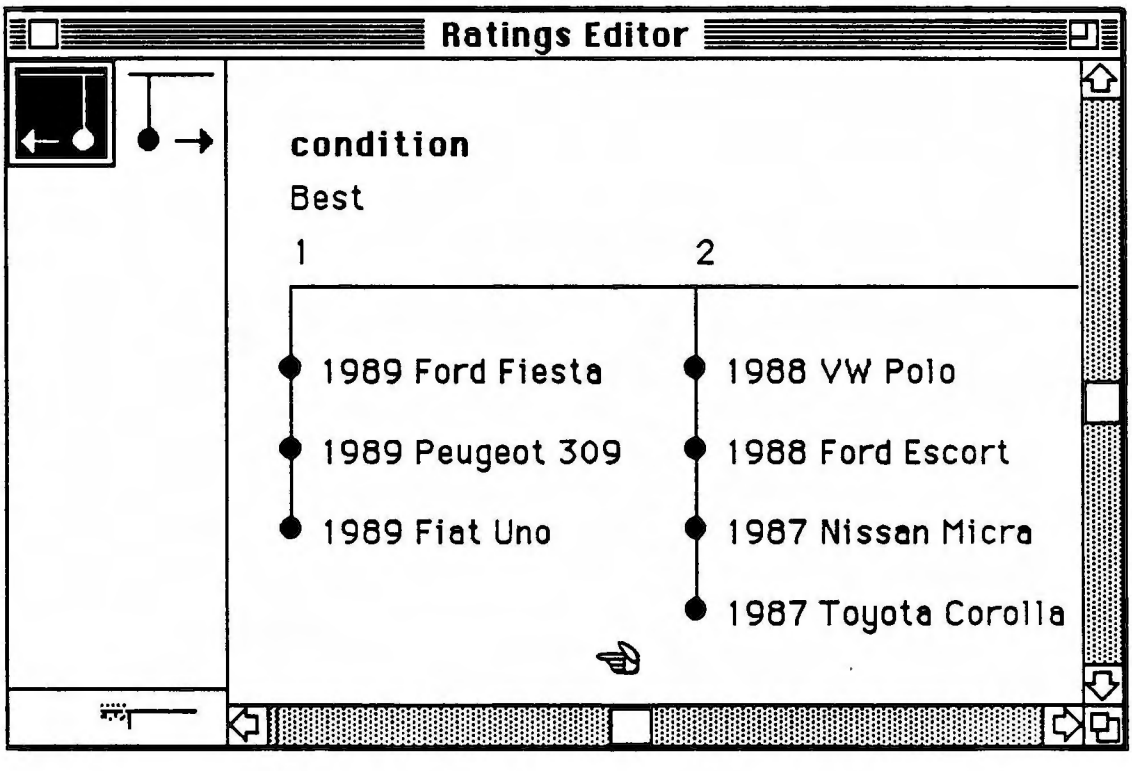


Figure 11. Increasing the precision of elicited ratings

In the case of a criterion for which the necessary data are either unavailable or unobtainable (for example because the attribute cannot be objectively measured), preferences can be expressed only in terms of a relative rating of the alternatives. The condition of a second-hand car, for example, is often a matter of subjective judgement. The user's ratings of alternatives with respect to such criteria are elicited [3] by a similar technique to the one described above for eliciting the preferred values of an attribute. As Figure 11 illustrates, the Ratings Editor provides interactive graphics tools for increasing the precision of elicited ratings.

6 Eliciting non-monotonic preferences

The expression of preferences in terms of a relative rating of alternatives is feasible only when the number of alternatives is small, and unnecessary when published data are available from consumer magazines or other sources. McSherry [3] describes how numeric criteria such as price can be incorporated in the reasoning process by weighting alternatives according to their values. Often it is sufficient to distinguish between criteria whose values the decision maker would wish to maximise (such as fuel consumption as measured in miles per gallon) and those whose values he or she would wish to minimise (such as price). In the latter case, alternatives are weighted according to the reciprocals of their values.

Not all decision criteria, however, are *preferentially monotonic* [3] in the sense that a higher value is always preferred to a lower one or *vice versa*. Often the decision maker will have in mind an ideal range of values, but would not necessarily wish to rule out alternatives which lie outside the limits of this ideal range. For example, the decision maker may prefer a value of engine size which is not too high and not too low. On the other hand, he or she may be indifferent to the value as long as it is not too low, say less than 1100 cc. Once again, the decision maker may not wish to rule out an alternative which is ideal in other respects but has an engine size of 1000 cc. As these examples illustrate, absolute constraints fail to capture the flexible approach to problem solving which MCDM often demands.

Figure 12 shows how non-monotonic preferences are modelled in McExpert. The preferred range of values for engine size (measured in cubic centimetres) is from 1100 to 1300. Values within the preferred range are given preference scores of 1. The minimum and maximum values of the attribute (among the available alternatives) are 900 and 1600. An engine size of 1100 is preferred four times more than an engine size of 900 which is therefore given a preference score of 4. Similarly, an engine size of 1300 is preferred three times more than an engine size of 1600. The preference score of an engine size

between 1300 and 1600 is obtained by linear interpolation between 1 and 3. For example, the preference score of an engine size of 1450 would be 2. Similarly, the preference score of an engine size between 900 and 1100 is obtained by linear interpolation between 4 and 1.

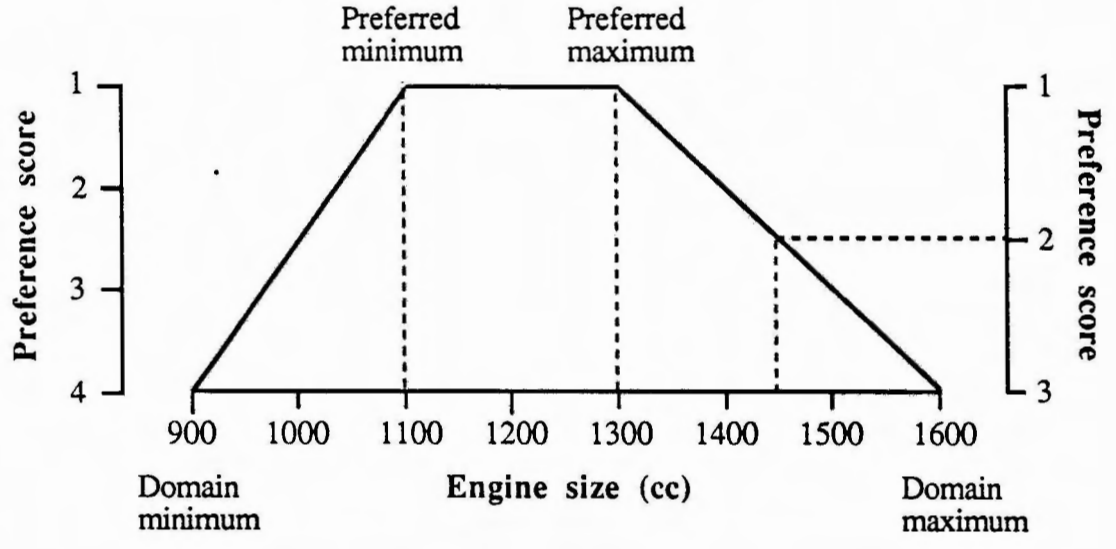


Figure 12. Modelling non-monotonic preferences

Figure 13 illustrates the first step in the elicitation of non-monotonic preferences in McExpert. The user is invited to indicate his or her attitude to the attribute. The first two options available to the user (the higher the better and the lower the better) indicate that the attribute is preferentially monotonic, in which case alternatives can be weighted according to their values or the reciprocals of their values. In the example of Figure 13, the user prefers an engine size which is not too high and not too low. Engine size is therefore preferentially non-monotonic (as far as he or she is concerned).

Figure 14 illustrates how the preferred range of values is elicited. The decision maker is invited to enter the preferred minimum and maximum engine sizes. He or she also has the option of entering the minimum and maximum possible values, which by default are set to the minimum and maximum values among all the alternatives in the knowledge base.

Finally, the decision maker is asked how much the preferred maximum value is preferred to the maximum possible value and (as Figure 15 illustrates) how much the preferred minimum value is preferred to the minimum possible value. The options available to the user correspond to the standard preference scores 1,3,5,7 and 9 of the

AHP. By selecting two adjacent radio buttons, the user may choose instead an intermediate preference score (such as 4 in the example of Figure 15).

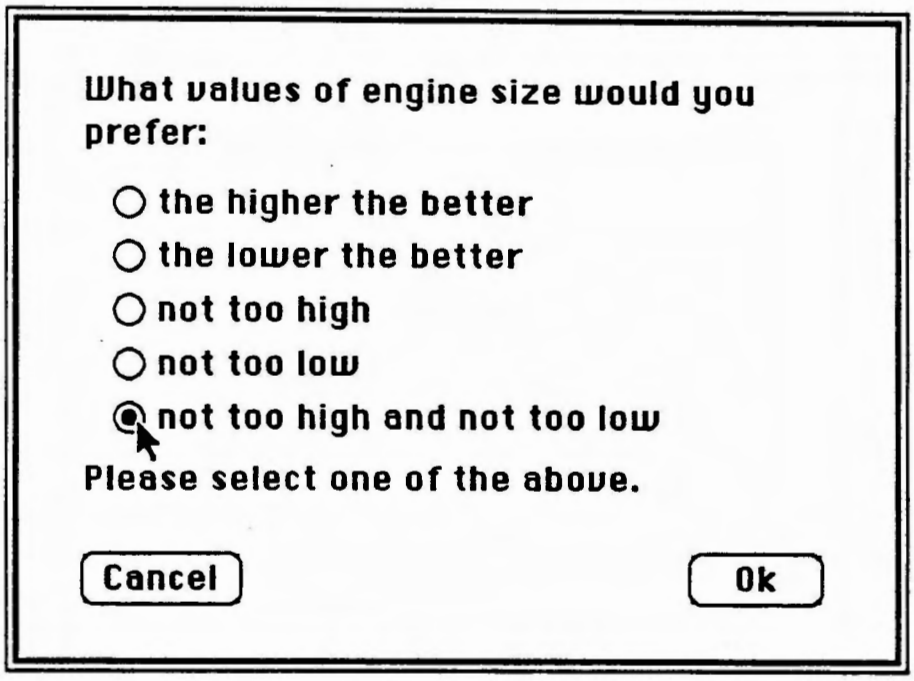


Figure 13. Eliciting the decision-maker's attitude to a numeric attribute

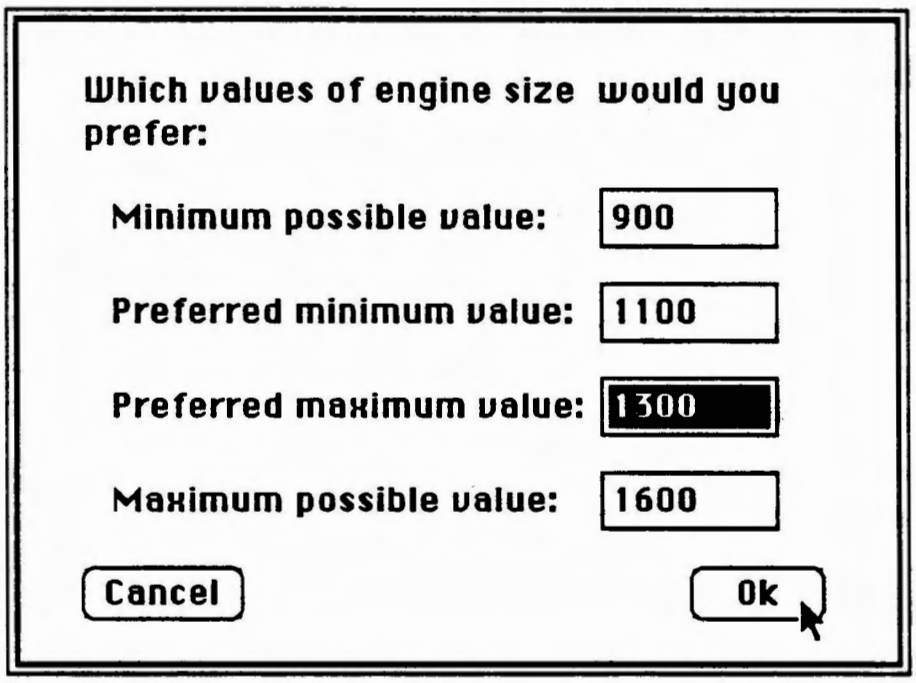


Figure 14. Eliciting the preferred minimum and maximum values

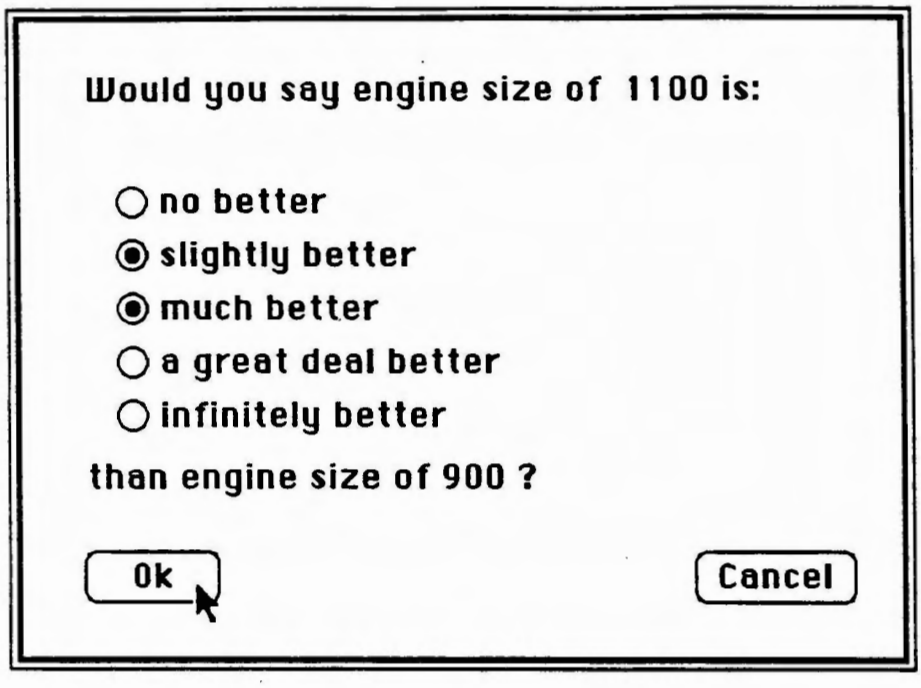


Figure 15. Eliciting a preference score for the domain minimum

In the case of an attribute for which the decision maker prefers a value which is not too high (or not too low), it is necessary only to elicit the preferred maximum (or minimum) value and how much that value is preferred to the maximum (or minimum) possible value.

7 The consultation environment

The aim of the consultation is to find the alternative which most closely matches the requirements of the decision maker. The first stage of the consultation is the elimination of alternatives which fail to satisfy any absolute constraints which have been elicited. The list of surviving alternatives is displayed for inspection by the user. Next, the decision-maker's priorities and preferences (unless elicited previously) are elicited. Finally, any case-specific information necessary for the evaluation of alternatives [3] is elicited. At the end of the consultation, the conclusions are displayed graphically in the form of a bar chart showing the overall ratings of the alternatives on a percentage basis. In the example of Figure 16, the elimination of alternatives which fail to satisfy absolute constraints specified by the user has reduced the number of alternatives to just two.

Rather than accepting the conclusions as final, the decision maker can use McExpert's knowledge elicitation tools for interactive sensitivity analysis. Using the Priorities Editor,

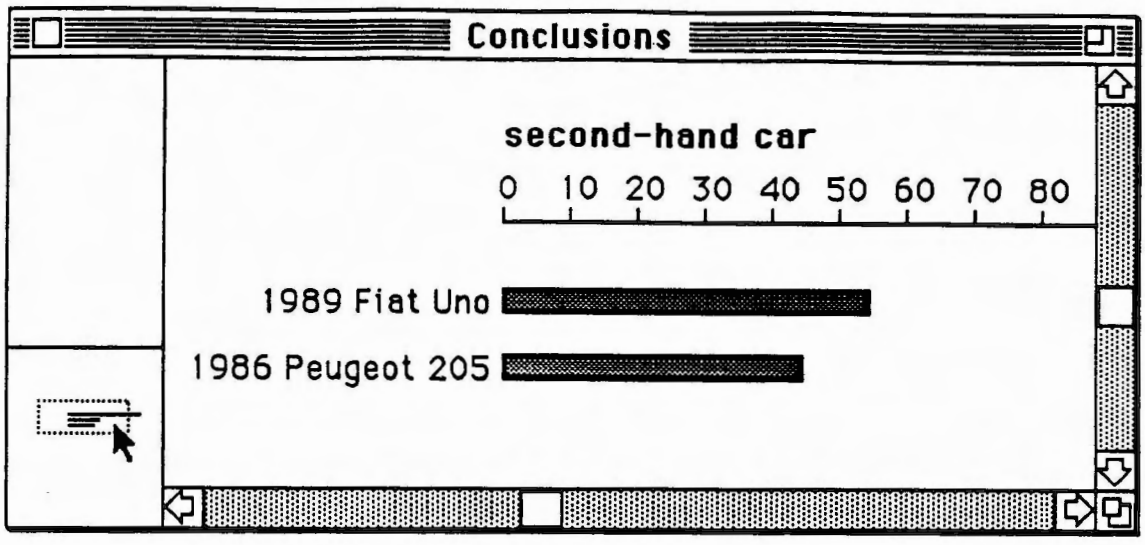


Figure 16. Overall percentage ratings of surviving alternatives based on the priorities and preferences of the user.

for example, he or she can examine the effects of fine tuning previously elicited priorities (Figure 17). The decision maker may also investigate the effects of introducing additional factors or alternatives. Sensitivity analysis may show the effects of such changes to be negligible, thus increasing his or her confidence in the overall results. On the other hand, it may reveal that factors or alternatives not previously considered would significantly affect the outcome.

Discussion

McExpert differs from other knowledge elicitation environments such as AQUINAS [2], its predecessor ETS, and KITTEN [7] in not relying on personal construct theory. Although the repertory grid [6] is a powerful technique for helping experts to identify important domain concepts in the early stages of knowledge elicitation, the elicitation of constructs by triads is a painstaking process compared with McExpert's modified AHP techniques for eliciting user priorities and preferences.

In McExpert, much of the knowledge is elicited directly from the decision maker at consultation time and not entirely in advance as in AQUINAS [2]. For repertory grid elicitation in AQUINAS and other environments, users enter numbers representing their ratings of alternatives. An exception is KITTEN, which provides interactive graphics tools for repertory grid elicitation [7]. In McExpert, decision makers can express their priorities and preferences (except in the case of non-monotonic preferences) qualitatively rather than quantitatively. If necessary, McExpert's interactive graphics tools can be used to increase the precision of the elicited priorities and preferences.

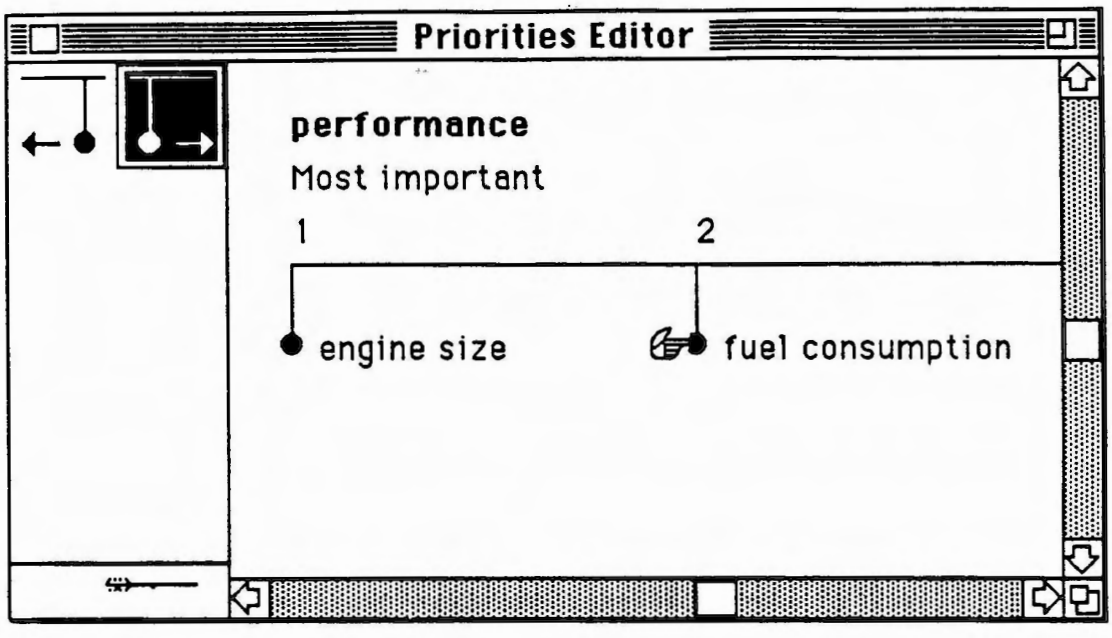


Figure 17. The Priorities Editor provides interactive graphics tools for sensitivity analysis

Given the inherent limitations of rule-based formalisms for MCDM [3], McExpert (unlike AQUINAS and similar environments) is not designed as a workbench for prototyping knowledge bases for rule-based expert system shells. Instead of generating rules, it combines KBS techniques with a logic programming adaptation of the AHP to provide decision-making support throughout the MCDM process.

This paper has described several enhancements of the tools and techniques provided in McExpert to support MCDM, including techniques for the elicitation of absolute constraints and non-monotonic preferences. While absolute constraints have an important role to play in reducing the number of alternatives to be considered, it has been argued here that they fail to capture the flexible approach to problem solving which MCDM often demands. The elicitation of non-monotonic preferences provides a more flexible alternative to absolute constraints by enabling alternatives which are considered less than ideal by the user to be weighted appropriately without eliminating them entirely from consideration.

References

[1] Bradshaw, J. M. and Boose, J. H. (1990). Decision analysis techniques for knowledge acquisition: combining information and preferences using Aquinas and Axotl. *International Journal of Man-Machine Studies*, 32:121-186.

[2] Boose, J. H. and Bradshaw, J. M. (1988). Expertise transfer and complex problems: using AQUINAS as a knowledge-acquisition workbench for knowledge-based systems. In Boose, J. H. and Gaines, B. R. (eds) *Knowledge acquisition tools for expert systems*, pp 39-64, Academic Press, London.

[3] McSherry, D. (1990). Knowledge-based systems for multi-criterion decision making. In Addis, T.R. and Muir, R.M. (eds) *Research and Development in Expert Systems IV*, pp 159-170, Cambridge University Press, Cambridge.

[4] McSherry, D. (1991). Modelling user priorities and preferences in a prototype expert system for the estate agent's office. *Proceedings of Informatics 11*. In Press.

[5] Saaty, T. L. (1980). *The analytical hierarchy process*, McGraw-Hill International, New York.

[6] Shaw, M.L.G. (1980). *On Becoming a Personal Scientist*, Academic Press, London.

[7] Shaw, M.L.G. and Gaines B.R. (1988) KITTEN: Knowledge initiation and transfer tools for experts and novices. In Boose, J. H. and Gaines, B. R. (eds) *Knowledge acquisition tools for expert systems*, pp 309-338, Academic Press, London.

Knowledge Acquisition Techniques and Second-Generation Expert Systems

MARIE-ODILE CORDIER
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex, FRANCE
e-mail : cordier@irisa.fr - phone : (33) 99 36 20 00
fax : (33) 16 99 38 38 32

CHANTAL REYNAUD
Laboratoire de Recherche en Informatique, Université Paris-Sud, Bât. 490, 91405 ORSAY Cedex, FRANCE
e-mail : cr@frlri61.bitnet - phone : (33) 16 1 69 41 64 95
fax : (33) 16 1 64 46 19 92

This paper presents techniques used in ADELE, a Second-Generation Expert System (SGES), to support the knowledge acquisition activity in the diagnostic domain. The approach has been studied inside the framework of SGES ; it is based on the relations between knowledge acquisition and explanations. When new heuristic knowledge is acquired, its justifications are looked for in domain models to support the knowledge acquisition process. ADELE is a medical diagnostic reasoning system for electromyography.

INTRODUCTION

Interest in the adoption of deep models in diagnostic expert systems has grown considerably in the last few years. Thus, two alternative approaches are proposed for modeling diagnostic problem solving.

The first one, the "model-based" diagnosis approach, uses models of the system to be diagnosed. The diagnosis results directly from a model-based reasoning. According to this viewpoint, the logical approaches deal with models of the correct behavior [Reiter, 1987] or fault models of the system to be diagnosed [Console et al., 1989]. Depending on the model used, different reasoning techniques have been developped. A diagnosis is a minimal set of abnormalities (assumptions about the abnormal behavior of some components of the system) which logically is consistent with or which explain the observations [Console et al., 1990].

The second approach uses heuristic knowledge (experience-based knowledge) to model the expert-like kind of reasoning. Each chunk of knowledge encodes associational or derivative links but justifications for this derivative process are not explicitly given.This experience-based

knowledge can be seen as a compiled form of the first approach as in [Chandrasekaran and Mittal, 1983] or as another form of knowledge, eventually more appropriate for diagnosis as in [David and Krivine, 1989] ; efficiency can be gained but in all cases the theoretical foundations of the deductive process are left out.

First-generation Expert Systems clearly belong to this second approach. Their limitations were recognized [Clancey, 1981] and therefore Second-Generation Expert Systems (SGES) try to use model-based knowledge and experience-based knowledge in order to profit from both approaches [Steels, 1985] [Davis, 1989]. Because SGES are more structured systems, important advantages lie in knowledge acquisition and explanations [Reynaud, 1989].

In this paper, we present techniques used in ADELE, a SGES, to support the knowledge acquisition activity. The problem of knowledge acquisition is not new. Finding heuristic knowledge (let us call them 'rules') has turned out to be extremely difficult. Experts typically take a long time to come up with solid rules, the rule-set seems never complete, is continously changing and shows inconsistencies between experts (and even within the same expert). The approach developed here is innovative because this problem is studied inside the framework of SGES in which more than one representation of the same domain and of the diagnostic problem is used.

Several approaches to knowledge acquisition inside the framework of SGES have been proposed in the last few years. Knowledge acquisition can be seen as a learning process. So, SGES can learn new rules by examining the results of deep reasoning [Steels, 1985]. In [Charlet, 1989], model-based knowledge is used to automatically produce production rules which are suggested to the expert who then decides whether to keep or to forget them. In KADS [Wielinga and Breuker, 1984], [Gardin and Minati, 1988] and [Abel, 1988], model-based knowledge is used to interactively acquire expert knowledge from a user, making the elicitation process easier.

In this paper, we present a different approach, as will be seen in I. When new heuristic knowledge (a production rule) is acquired, abductive reasoning based on the model-based knowledge provide justifications which can be regarded as proofs of the association between the conditions and the conclusion of the rule. These justifications are useful to support the knowledge acquisition process : they can be used to propose explanations, to comment on the rule, to control it, to suggest modifications or other rules.

ADELE is a medical diagnostic reasoning system for electromyography, the diagnosis of muscles and peripheral nerves disorders by electrical measurements. Examples in this field will

be used in the paper. Electromyography is also the application domain of the project P599 supported in part by the EEC Esprit Program [Andreassen and al., 1987]. This work uses a causal probabilistic network for interpretation of Electromyographic findings : MUNIN. It has been interested in knowledge acquisition too but particularly in the way of estimating conditional probabilities. The applied method restricts the possible choices of probabilities by using "deep knowledge" i.e. models of pathophysiological processes as expressed in medical textbooks and papers.

The paper is organized as follows. We first give a general description of our approach. In section II, we describe the model-based knowledge. Then we outline how the model-based knowledge is used to compute explanations for heuristic knowledge. In section IV, appropriate criteria for analysing explanations are identified whilst the last section focuses on the way the knowledge acquisition activity can take advantage of this analysis.

I THE KNOWLEDGE ACQUISITION APPROACH IN ADELE

ADELE is composed of two knowledge bases : the experience-based knowledge base noted RB (or *Rule Base*) and the model-based knowledge base noted KB (or *Knowledge Base*).

RB is composed of production rules[1] (Antecedents, Conclusions) denoted (Ant, Cl) where Ant = ant1, ant2, ..., antn and Cl = cl1, cl2, ..., clm. For simplicity we will consider the case where m = 1. A rule encodes an association between antecedents and conclusion which is purely operational. It suggests that "if antecedents belong to the fact base, then conclusion can be added to the fact base". Production rules are shortcuts of an expert reasoning. The record of inference steps that ties antecedents to conclusion has been left out. Rules don't mention the underlying process upon which they are based as in the well-known example "if the patient is less than 8 years old, don't prescribe tetracycline" [Clancey, 1981]. So frequently, we refer to rules as 'compiled knowledge' and as knowledge which contains tacit knowledge, sometimes interesting to explain.

Knowledge acquisition techniques are used in ADELE when the expert gives a production rule, R:(Ant,Cl), encoding some chunk of heuristic knowledge. Each production rule is considered separately.The objective is to try to prove the association existing between Ant and Cl or between Cl and Ant, with the model-based knowledge base, KB. The main function of KB is to produce some proof(s) of R:(Ant,Cl).

---

[1] We consider here the restricted case where a rule does not contain any variables or quantifiers.

This can be formalized so :

For each rule R:(Ant, Cl), try to prove :

KB, Ant I== Cl

or

KB, Cl I== Ant.

KB is represented with a formalism closely related to the semantic network formalism whose semantics is related to the graph theory. Getting proofs of R:(Ant,Cl) doesn't use the logical approach but it rather concerns getting paths in a semantic network. A proof of R:(Ant, Cl) is a sub-graph of KB.

The proofs or justifications of R:(Ant,Cl) are the basic material for knowledge acquisition techniques. We argued that a precise analysis of these explanations produce interesting results on the nature of the link between conditions and conclusions, the strength of this link, the roles played by the conditions and the discriminating power of the conditions over the conclusion. The acquisition module will take advantage of this analysis to explain the rule, comment on it, check it, suggest modifications or other rules.

The Experience-based Knowledge Base of ADELE, applied to medical diagnosis, was built in cooperation with the INSERM group. It was used as a diagnostic tool and has been validated. 95 production rules have been written. A complete description of the project can be found in [Reynaud, 1989]. Subsequently, we will use the following simple rule relative to this diagnosis to illustrate the method : "if Paresthesie in the median distal region then Carpal channel syndrome". It expresses the relation between the symptom 'Paresthesie' (prickling) confined to a part of the arm and the disease 'Carpal channel syndrome' being an obstruction of the channel of the median nerve at the wrist.

II THE MODEL-BASED KNOWLEDGE (KB)

KB is represented with a formalism closely related to the semantic network formalism. As we want to have some advantages of abstraction without losing any information, the domain modelization has been made on two levels and has given rise to two kinds of models.

The Instantiated Model

The semantic network restricted to objects and relations between objects is termed *the Instantiated Model*. These objects refer directly to the antecedents and conclusion of the production rules of RB.

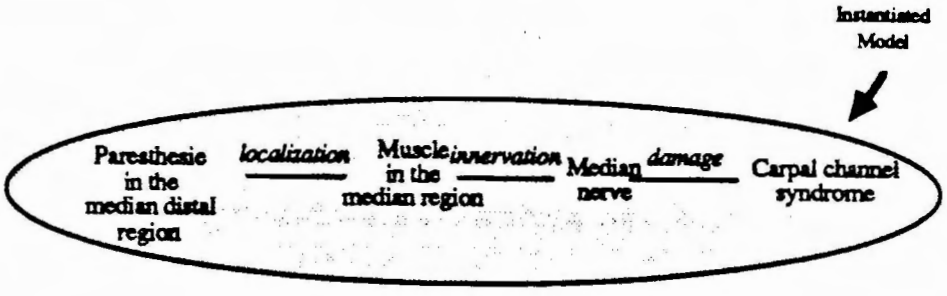


Figure 1

The Conceptual Model

The semantic network restricted to classes of objects and relations between classes of objects is termed *the Conceptual Model.*

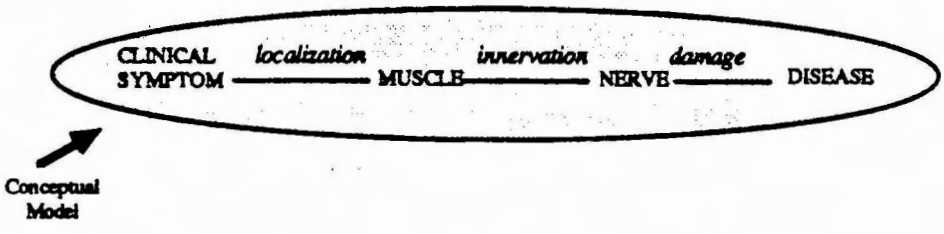


Figure 2

How are the relations of the models described ?

A relation R(X,Y) is described by :

- its name : the name of the relation will be used to describe, in a litteral form, links between objects or classes of objects ; a relation corresponds to two litteral forms, one relating X to Y and one relating Y to X.
  innervation is a relation between NERVES and MUSCLES ; it can be described by "a MUSCLE *is innervated by* a NERVE" or by "a NERVE *innervates* a MUSCLE".
- its type (or view point) which can be : neuro-anatomical, pathophysiological, clinical... and denotes the type of domain knowledge that the relation refers to.
- its nature : a relation can be hierarchic (the relation *is-a* between an object and its class), descriptive (the relation *localization*), functional (the relation *innervation*), or causal/evocative (the relation *damage/is damaged by*). A causal relation needs to be oriented. It will be causal or evocative according to its orientation.

Example 1

- the relation *innervation* between NERVES and MUSCLES corresponds to an anatomical view point ; it is a descriptive relation.
- the relation between a DISEASE and the EXPECTED CHANGES ON NERVES corresponds to a pathophysiological view point ; it is a causal relation.
- the relation between a DISEASE and PRESENCE OF CLINICAL SYMPTOMS corresponds to a clinical view point ; it is a weak causal relation.

How to build these models ?

Instantiated Models model objects and relations between objects of the domain. These objects refer directly to the antecedents and conclusions of the production rules of RB.

Conceptual Models are obtained from Instantiated Models. Classes of objects stem from grouping together objects and relations between classes of objects stem from generalizing relations between objects. We make the following assumption :

Assumption 1: if there exists a relation R between two objects, there exists a relation R between the classes that the objects belong to.

Relations between the Conceptual Model and the Instantiated Model are exclusively *is-a* relations between classes of objects and objects which belong to them (see figure 3).

The structure of KB is very interesting both from a methodological view point and from a computational one.

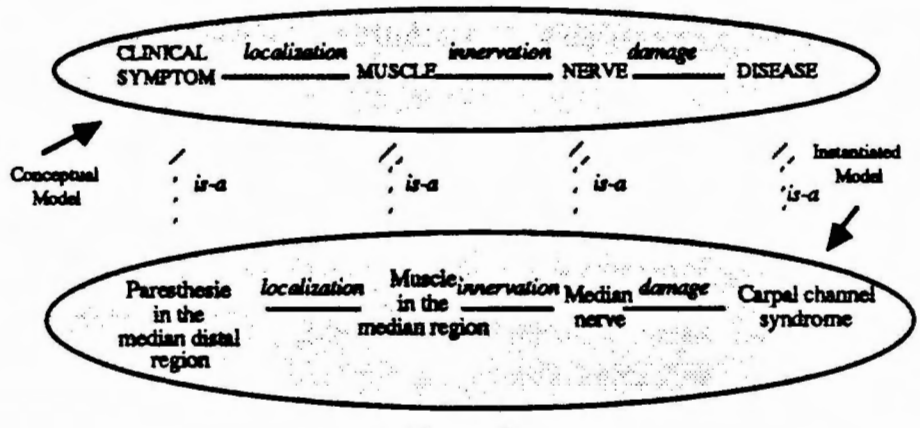
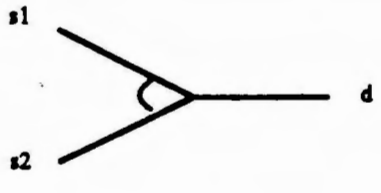


Figure 3

The diagnostic model described here is not a pure causal model. Causal relations coexist with descriptive relations. Thus the diagnostic model is not unique. A model depends on the author and on its special field. Therefore, KB can contain a variety of models which refer to multiple view points and which can be divided into Instantiated and Conceptual Models.

Sometimes, we need to represent two objects (or classes of objects) which are in relation with a third one. For example, a symptom s_1 and a symptom s_2 are in relation with a disease d because s_1 is always associated with s_2 when the disease d appears. This can be represented so :



For this reason, KB can be seen as an AND/OR graph or more exactly a hypergraph. Each relation R(X,Y) is a relation between X and Y where we can have X = $\{x_1,x_2\}$ and Y = $\{y\}$. All sub-graphs of KB related in this paper are in fact sub-hypergraphs of KB.

In most of the examples of the paper, X and Y will be reduced to one element.

III THE EXPLANATIONS

As seen in I, production rules do not make explicit the semantic link between conditions and conclusions. When a rule R:(Ant,Cl) is added to RB, Explanations (or justifications) for R:(Ant,Cl) are searched in the model-based knowledge base KB to make it explicit. Explanations of a rule refer to Explanations of a pair (a,b) as described below.

Explanations of a pair (a,b)

We essentially consider Explanations of a pair (a,b) where a = $\{a_i, ..., a_n\}$ and b is a singleton. E will be an Explanation of (a,b) iff E is a sub-graph G such that :

$\forall\ a_i \in a$ then $a_i \in G$
$b \in G$
$\forall\ a_i \in a$, a path exists between a_i and b.

A path between two objects contain only objects. A path between two classes of objects contain only classes of objects.

A path exists between X = $\{x_1, ..., x_n\}$ and Y = $\{y_1, ..., y_n\}$ iff :

we have a relation R between X and Y with R ≠ is-a
or
we have a node Z such that :
  we have a relation between X and Z
  and
  a path between Z and Y.

An Explanation of a pair (a,b) where a and b are objects will be called *an instantiated Explanation* ; an Explanation of a pair (A,B) where A and B are classes of objects will be called *a conceptual Explanation.*

Abstraction of an (instantiated) Explanation of a pair (a,b) (conceptualization)

An abstraction of an instantiated Explanation of a pair (a,b) is a (*conceptual*) Explanation of (A,B) where :

A is a set of classes of objects, B is a class of objects
for a = $\{a_i\}$, A = $\{A_i$ / is-a $(a_i,A)\}$
is-a (b,B).

It is a sub-graph G such that $\forall A_i$, $A_i \in G$, $B \in G$, $\forall A_i \in A$, a path exists between A_i and B.

It will be called *Conceptual Explanation for (a,b).*

Instance of a (Conceptual) Explanation of a pair (A,B)

An instance of a (conceptual) Explanation G of a pair (A,B) is an instantiated Explanation F such that G is an abstraction of F.

Instances of the same conceptual Explanation are called *Brother-Explanations.*

Assumption 2 : If E is an Explanation for (a, b), there exists a Conceptual Explanation for (a, b) (see assumption 1), but an instance of a Conceptual Explanation for (a, b) is generally not an Explanation for (a, b).

Assumption 3 : if E is an instance of a Conceptual Explanation for (a, b) F, such that $\forall a_i \in a$, $a_i \in E$ and $b \in E$, then E is an Explanation for (a, b).

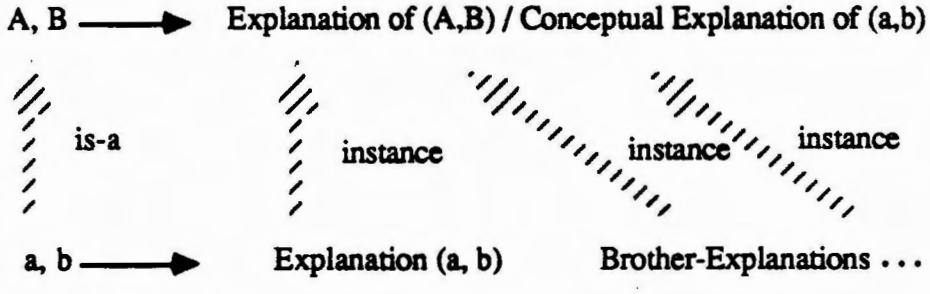


Figure 4

Exemple 2

Figure 5 below is an Explanation F[2] between DISEASE and CLINICAL SYMPTOM. It suggests that :

"a DISEASE modifies the ELECTRICAL MEASUREMENT FROM A NERVE ; An ELECTRICAL MEASUREMENT FROM A NERVE depends on an ELECTRICAL MEASUREMENT and on a NERVE. NERVES are damaged by DISEASES. a NERVE innervates a MUSCLE ; a MUSCLE is a place where CLINICAL SYMPTOMS are localized. Figures 6, 7 and 8 are instances of F.

[2] In fact, the links between nodes are represented by arcs and not lines. Yet, at first, we will use Explanations without taking direction into account. So, this one doesn't appear on the examples.

A (conceptual) Explanation G between DISEASE and CLINICAL SYMPTOM

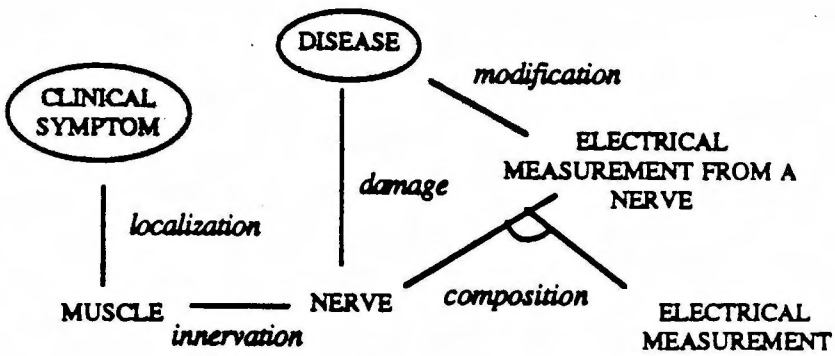


Figure 5

Instances of the the (conceptual) Explanation G between DISEASE and CLINICAL SYMPTOM

Instance 1 :

Carpal channel syndrome modifies Amplitude sensitivity of the median nerve ; Amplitude sensitivity of the median nerve depends on Amplitude sensitivity and on the Median nerve ; the Median nerve innervates the Muscle in the median nerve ; the Muscle in the median nerve is a place where Paresthesie in the median distal region is localized.

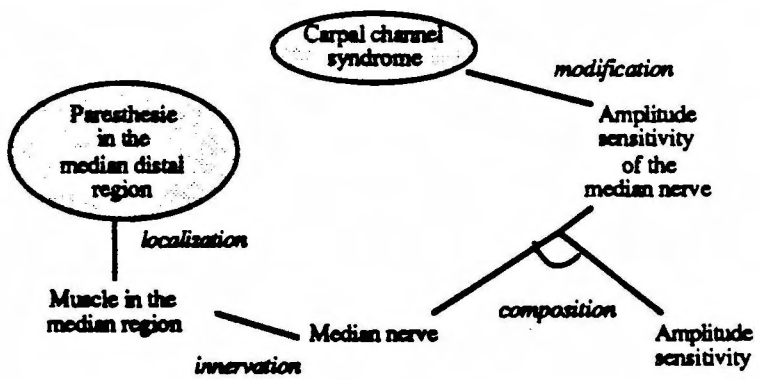


Figure 6

Instance 2 :

Carpal channel syndrome damages the Median nerve ; the Median nerve innervates the Muscle in the median nerve ; the Muscle in the median nerve is a place where Paresthesie in the median distal region is localized.

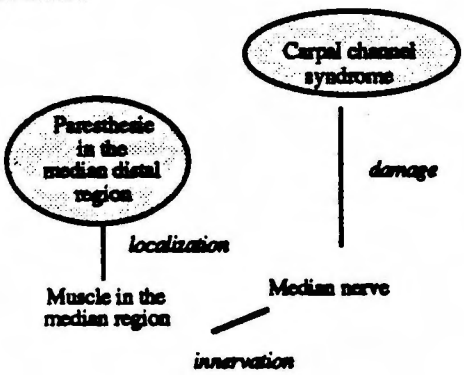


Figure 7

Instance 3 :
Carpal channel syndrome damages the Median nerve ; the Median nerve innervates the Muscle in the median nerve ; the Muscle in the median nerve is a place where Hypoesthesie in the median region is localized.

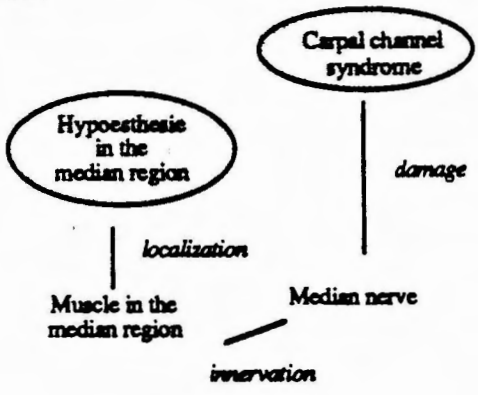


Figure 8

Explanations for a rule R:(Ant,Cl)

E will be an Explanation for a rule R:(Ant, Cl) iff E is an Explanation for (Ant',Cl)[3] with Ant' $\subseteq$ Ant. Explanations for (Ant,Cl) are obtained by looking for Conceptual Explanations for (Ant,Cl) and then trying to find suitable instances.

For example, to explain the rule "if Paresthesie in the median distal region then Carpal channel syndrome", we are looking for an Explanation for (Paresthesie in the median distal region, Carpal channel syndrome). It will be obtained by looking for a (conceptual) Explanation for (A,B) : (CLINICAL SYMPTOM, DISEASE). We obtain the conceptual Explanation described in figure 5. Its instances are there calculated : see figure 6, figure 7, figure 8. Among these, only the Explanation shown in figure 6 is retained as a suitable Explanation for (Paresthesie in the median distal region, Carpal channel syndrome).

IV ANALYSIS OF EXPLANATIONS

These "brute" Explanations are the basic material for acquisition; but it is supposed that a precise analysis of these Explanations can produce interesting results by following : the nature of the link between conditions and conclusions, the strength of this link, the roles played by the conditions, the discriminating power of the conditions over the conclusion. The acquisition module will exploit this analysis to explain the rule, comment on it, check it, suggest modifications or other rules as it will be seen in the last section.

---

[3] The experience-based knowledge base RB and the model-based KB do not share the same representation language. When a rule R composed of its antecedents Ant and its conclusion Cl is added to RB, R has then to be translated into a pair (a,b) where a results from the translation of Ant and b from the translation of Cl. This translation process is not detailed in this paper.

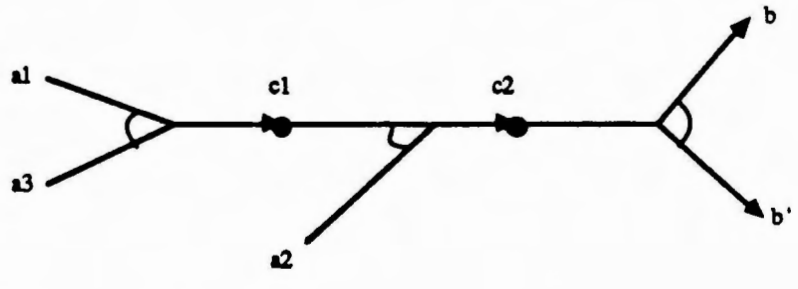
In this section, criteria for analysing Explanations are defined and illustrated. In the next section, it will be shown how these criteria are used in the acquisition process.

The criteria for analysing a rule can be divided into two types : criteria for analysing an Explanation per se, or criteria for analysing a set of Explanations. Before explaining in detail these criteria, we need to define what we call *an oriented Explanation*. A rule is oriented from conditions to conclusions. Relations between objects in an Explanation have not, until now, been considered as oriented relations ; now we need to explain how an Explanation can be oriented to "mimic" more closely a rule.

Oriented Explanation

An Explanation is oriented by labelling the nodes of an Explanation : the nodes corresponding to conditions are first labelled as sources. By propagation of markers, an algorithm divides the nodes of the Explanation in three classes : O : the set of sources, I : the set of intermediate nodes and X : the set of extremities. For an oriented Explanation of a rule R(Ant,Cl), we have : Ant $\subseteq$ O and Cl $\subseteq$ X.

Example 3 : an Explanation for ($\{a_1, a_2\}$, b)



$O = \{a_1, a_2, a_3\}$ $I = \{c_1, c_2\}$ $X = \{b, b'\}$

Figure 9

Criteria for Analysing an Explanation

1) Nature of the Explanation : the nature of the Explanation can be used to evaluate the strength of the deductive link of a rule. It results from the natures of the composing relations : descriptive, functional, causal or evocative. In an oriented Explanation, it is possible to decide if a relation is causal (going from causes to effects) or evocative (going from effects to causes). An Explanation is said to be causal if the relations are descriptive, functional or causal; evocative if the relations are descriptive or causal or evocative; if neither then undefined.

Example 4

The Explanation below is an evocative one for "if Paresthesie in the median distal region then Carpal channel syndrome".

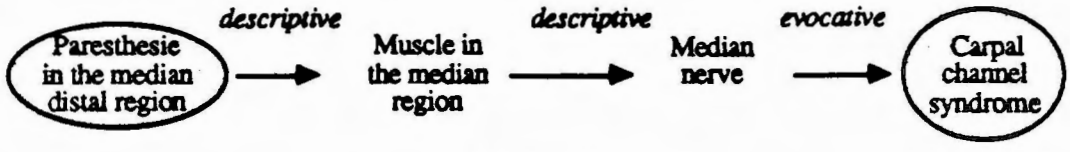


Figure 10

The Explanation for "if Carpal channel syndrome then Paresthesie in the median distal region" will be a causal one as can be seen below :

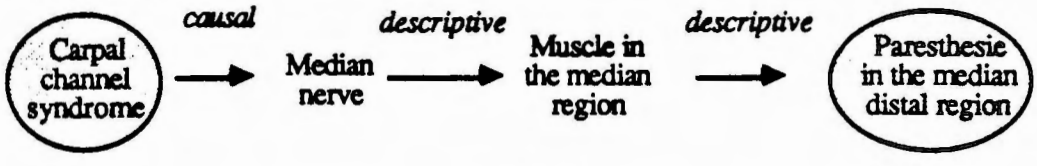


Figure 11

2) Coverage of a Rule by an Explanation : a rule R(Ant,Cl) is *α-covered by an Explanation* E iff :

- E is an Explanation for (α, Cl) with $\alpha \subseteq$ Ant
- $\forall$ Ant", $\alpha \subset$ Ant", E is not an Explanation for (Ant", Cl).

An Explanation E is said to be *total* for R:(Ant,Cl) iff R is Ant-covered by E; an Explanation E is said to be *partial* for R:(Ant,Cl) iff R is α-covered by E and $\alpha \subset$ Ant.

This criteria, as for the next one, is useful to analyse the conditions of a rule and the strength of the link between conditions and conclusion of a rule.

Example 5

The Explanation of figure 10 is partial for the rule "if Paresthesie in the median distal region and Woman then Carpal channel syndrome" because "Woman", a factor in favour of the disease, isn't covered.

3) Adequacy of an Explanation to a Rule : the set O of sources of an Explanation can contain nodes which do not correspond to conditions of the explained rule R:(Ant,Cl). The set {O - {Ant}} contains the objects (or classes of objects) which are not explicitly stated as conditions in the rule but are required to explain the conclusion. When this set is empty, it means that the conditions are sufficient to explain the conclusion; if not, it means that supplementary conditions are necessary to explain the conclusion.

Example 6

The Explanation represented on figure 9 is adequate for a rule whose conditions are $\{a_1, a_2, a_3\}$ but inadequate for a rule whose conditions are $\{a_1, a_2\}$.

4) Brother-Explanations : They are used to suggest other rules. Let E, the analysed Explanation, be an Explanation for (Ant,Cl) :

- if E_1 is an Explanation for (Ant,Cl') and E_1 is a brother-Explanation of E, it means that E_1 can explain a distinct conclusion from the same conditions with the same scheme of Explanation ;
- if E_2 is an Explanation for (Ant',Cl) and E_2 is a brother-Explanation of E, it means that E_2 can explain the same conclusion from a set of distinct conditions with the same scheme of Explanation.

Criteria for Analysing a set of Explanations

1) Coverage of a rule by a set of Explanations : using the concept of coverage of a rule by an Explanation, it is possible to define the coverage of a set of Explanations $\{E_i\}$. Let us note α_i the coverage of an Explanation E_i. The coverage of $\{E_i\}$ is $\cup\alpha_i$. A rule can then be totally or partially covered by a set of Explanations.

We can distinguish between :

- conditions covered by all the Explanations of the rule (central conditions),
- conditions not covered by any Explanations (un-explainable conditions),
- others (partially-explainable conditions).

2) Complementary Explanations : E_1 and E_2 are two complementary Explanations if the coverage of E_1 for R:(Ant,Cl) is α, the coverage of E_2 is β, and we have : $\alpha \subset \alpha \cup \beta$ and $\beta \subset \alpha \cup \beta$. The coverage of both is greater than the coverage of each of them. E_1 and E_2 are totally complementary iff $\alpha \cup \beta$ = Ant.

3) Comparable Explanations : E_1 and E_2 are two comparable Explanations if they have the same coverage for R:(Ant,Cl).

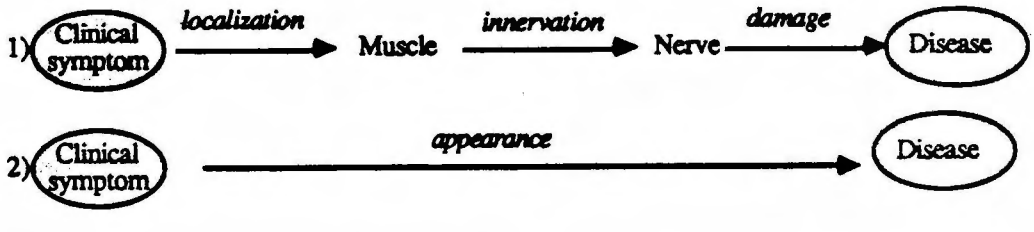
4) Refined Explanations : E_2 is said to be a refinement of an Explanation E_1 iff

- they have the same sources and the same extremities
- $\exists$ R(X, Y) $\in E_1$, $\exists$ $r_1, r_2, \ldots r_p \in E_2$, X, Y $\in E_1$ and X, Y$\in E_2$, and E_2 is obtained by replacing in E_1 R(X, Y) by :

$r_1 (X, Z_2) \circ r_2 (Z_2, Z_3) \circ \ldots \circ r_p (Z_p, Y)$.

A refined relation is transitive.

Example 7



The first Explanation is refined, the second is more general.

V USE OF EXPLANATIONS IN THE ACQUISITION PHASE

The knowledge acquisition activity can take advantage of the analysis of Explanations to check a rule, to justify it, to analyse it, to suggest other rules. All these points are detailed in this section.

To check a rule

The existence or non existence of Explanations can be used to check a rule and suggest errors in its formalization :

- Absence of Conceptual Explanations (and then of Instantiated Explanations) means that no link can be found in the model between any condition and the conclusion. See example 8 below.

Example 8

There is no explanations for "if Paresthesie in the median distal region then Cardal channel syndrome" because a spelling mistake has been made.

- Existence of a Conceptual Explanation and absence of Instantiated Explanations means that a link has been found between the classes of objects representing the conditions and the conclusion, but this link cannot be instantiated as it is made in the rule. See example 9 below.

It is important to remember that the model-based knowledge is not considered complete and that experience-based knowledge is not merely a compiled form of the model-based knowledge : absence of Explanations does not mean that the rule is false, but that the rule could be false and must be checked. Errors like those in example 9 can easily be detected and corrected (suggestions can then be made by the system as explained further).

Example 9

For "if Amplitude sensitivity of the cubital nerve = reduced then Carpal channel syndrome", a conceptual Explanation exists but there is no instantiated explanation. An instantiated Explanation would be obtained with "Amplitude sensitivity of **the median nerve** = reduced".

To justify a rule

Explanations of a rule are justifications that make explicit the semantic link existing between conditions and a conclusion. ADELE can propose different Explanations for a rule and for each of these Explanations can propose Conceptual Explanations or Instantiated Explanations. One of the problems is then to decide what Explanations are most appropriate to the user. Some of the criteria seen in the previous section will help to make this choice.

- the type (or view point) of the Explanation,
- the coverage of explanations : the best covering Explanations are often the most appropriate,
- complementary Explanations : in order to cover all the conditions of a rule,

- the degree of detail of an Explanation : it seems natural to begin by giving general Explanations before giving refined ones .

The solution adopted in ADELE is to propose a menu to the user who can then decide to display one or all Explanations or to select detailed or general Explanations. In all cases, general Explanations are given first. It is clear that this problem is a classical problem of man-machine communication and a user model is needed to decide which is the best choice.

To analyse a rule

Analysis of Explanations for a rule R:(Ant,Cl) can provide interesting information concerning it.

1) Conditions of the rule

- Non-explainable conditions : they are conditions which are not covered by any Explanations. They are often conditions which reflect heuristic context, but do not belong to the model.

Example 10
"if *Post-christmas period* and .... then suspect .....".
"if *Pregnancy* and Paresthesie in the median distal region then Carpal channel syndrome".
The two conditions which are written in italic style cannot be explained by models.

- Reinforcing conditions : when the adequacy of the Explanation is not total (see previous section). They correspond to nodes of an Explanation which are sources in the oriented Explanation but are not conditions in the rule. These objects participate in the Explanation but have been "forgotten" in the rule. Adding them as conditions will restrict the applicability of the rule, but will increase the strength of the deductive link existing between conditions and conclusion.
  If these objects participate in all the Explanations of a rule, there is a strong possibility that they are necessary and "forgotten" conditions.

2) Strength of the deductive link

Oriented Explanations can give an idea of the nature of the deductive link existing between conditions and conclusion. They can be helpful to evaluate the strength or the certainty of the deduction, especially when certainty factors are needed.

- the strength of the deduction can be evaluated from the nature of the elementary relations: a causal link is stronger than an evocative link; a definitional link is stronger.
- the strength of the deduction can be evaluated by looking at the reinforcing conditions.
- the strength of the deduction can be evaluated by considering brother-Explanations. The existence of brother-Explanations of the conclusion, i.e. explanations for (Ant, Cl') with

Cl ≠ Cl', reveals that other conclusions can be explained by the same conditions and following the same explicative scheme. When the nature of the link is evocative, it means that Ant is a symptom (or a set of symptoms) of Cl, but also of Cl', Cl", etc. The discriminating power of Ant for Cl is clearly inversely proportional to the number of brother-type Explanations. When the nature of the link is causal, Brother-Explanations of the conclusion can be used in the same way.

To suggest variant rules

Brother-Explanations can be used to suggest variant rules. These suggested rules are analogous to the analysed rule in the sense that they have exactly the same Conceptual Explanation but with different instantiations.

Brother-Explanations for conditions are Explanations of the conclusion by an other set of conditions (following the same scheme of Explanation).

Example 11

Analysing R : "if Paresthesie in the median distal region then Carpal channel syndrome" suggests this one :"if Hypoesthesie in the median distal region then Carpal channel syndrome".

Brother-Explanations of the conclusion are Explanations of other conclusions by the same set of conditions (following the same scheme of Explanation).

Example 12

Analysing R : "if Amplitude sensitivity of the Median nerve = reduced then Carpal channel syndrome" suggests this one : "if Amplitude sensitivity of the median nerve = reduced then Axonal suffering".

DISCUSSION AND CONCLUSION

In discussion, we will focus your attention on a few specific points. Firstly, such techniques can be used only when domain models are available as regards structure, function or causality. A typical example for which various models can be used is technical systems such as vehicles, electronic devices, etc. They are artefacts constructed to perform a particular function. All components are known and the behavior of the whole can theorically be predicted from the behavior of the parts. Another example (which is the one we are using in our experiments) is the medecine domain. Here, structural and functional models would be too complex but, nevertheless a lot of information about dysfunction are listed in medical textbooks or papers and can easily be represented with causal models. Secondly, our approach is based on schematic models which describe the domain entities and their relations. These models are descriptive or causal and represent one semantic of the domain. Using such models can support knowledge acquisition activity but we must be aware that the results we get have to be viewed as helpful comments or warnings and no more. According to these two points, our approach is

applied with all diagnostic problems. In particular, justifications can be required for all rules which evoke hypotheses i.e. troubles, badly working components or diseases from observations of abnormalities or symptoms. Such rules are important because they control the system's reasoning and specify what hypotheses are most likely to be problem solutions. Their derivative process is far from being clear. With justifications, we can control them and thus anyone will be able to understand what they mean. Note that such explanations can also be very beneficial when teaching students rules. It is much easier to memorize a rule when we are aware of its underlying derivative process.

The goal of this paper was to show how a structured diagnostic model can be used to support heuristic knowledge. In so doing we have shown how Explanations for heuristic rules are built from deep models, and how these Explanations can be analysed according to a set of criteria defined given a list in II. In ADELE, they are used to support knowledge acquisition ; they could also be used for diagnostics but in this paper we have only dealt with knowledge acquisition.

Knowledge Acquisition and Explanations obviously benefit from the explicit representation of domain "knowledge level" models. Other benefits would be obtained with an explicit representation of the conceptual model of the expertise : i.e. domain models and also resolving-problem models. Further work will be done in this direction.

REFERENCES

[Abel, 1988] Abel J., Abel C. 1988. Une méthodologie pour le développement des systèmes experts s'appuyant sur la notion de modèles sous-jacent. Journées Internationales sur les systèmes experts et leurs applications, Avignon.

[Andreassen and al., 1987] Andreassen S., Woldbye M., Falck B., Andersen S. 1987. MUNIN - A Causal Probabilistic Network for Interpretation of Electromyographic Findings, Proc. of the IJCAI conference, pp. 366-372, Milan.

[Chandrasekaran and Mittal, 1983] Chandrasekaran B., Mittal S. 1983. Deep versus Compiled Knowledge Approaches to Diagnostic Problem-solving. In Int. J. on Man-Machine Studies 19, pp. 425-436.

[Charlet, 1989] Charlet J. 1989. Lézard : Acquisition des Connaissances et Gestion de l'Incertitude dans un Système Expert de Seconde Génération, Thèse de doctorat, Université de Paris 6.

[Clancey, 1981] Clancey W. 1981. The Epistemology of a Rule-based Expert System : a Framework for Explanation, Artificial Intelligence, 20(3), pp. 215-251.

[Console and al., 1989] Console L., Theseider Dupre D., Torasso P. 1989. A Theory of Diagnosis for Incomplete Causal Models, in Proc. of the 11th IJCAI, pp. 1311-1317, Detroit.

[Console et al., 1990] Console L., Torasso P. 1990. Integrating Models of the Correct Behavior into Abductive Diagnosis, in the Proc. of ECAI, pp. 160-166, Stockholm.

[David and Krivine, 1989] David J. M., Krivine J. P. 1989. Designing Knowledge-Based Systems within Functional Architectures : the DIVA Experiment. In proc. of the Fifth IEEE Conference on Applications of Artificial Intelligence, Miami.

[Davis, 1989] Davis R., 1989. Form and Content in Model-Based Reasoning, Workshop : Model-Based Reasoning, Detroit.

[Gardin and Minati, 1988] Gardin F., Minati M. 1988. Deep Model for Knowledge Acquisition in Diagnostics, Proc. of IMACS.

[Reiter, 1987] A Theory of Diagnosis from first Principles, 1987. Artificial Intelligence, 32, pp. 57-95.

[Reynaud, 1989] Reynaud C. 1989. ADELE, un outil pour l'acquisition des connaissances basé sur des justifications, Thèse de doctorat, Université de Paris-Sud.

[Steels, 1985] Steels L. 1985. Second-Generation Expert Systems. Journal of Future Generation Computer Science 1, 4.

[Wielinga and Breuker, 1984] Wielinga B.J., Breuker J.A. 1984. Interpretation of verbal data for knowledge acquisition, Proc. of the ECAI conference, pp.41-50, Amsterdam.

Knowledge Acquisition in Context for a Complex Domain

Ashwin Srinivasan[†]
*Paul Compton**
Ron Malor
Glenn Edwards
Leslie Lazarus A.O.

Department of Chemical Pathology
St Vincent's Hospital
Darlinghurst NSW
Australia 2010

and

* School of Computer Science and Engineering
University of New South Wales
PO Box 1, Kensington NSW
Australia 2033

ABSTRACT

The knowledge provided by experts is always in a specific context and is essentially a justification of the expert's judgement in that context. Further, this justification is an identification of important differences in data between cases for which the judgement is correct and cases for which another judgement could be considered in the context. A knowledge acquisition methodology has been developed based on using knowledge only in the context in which it is provided and asking the expert to identify which differences in a list of possible differences provide a good justification. This paper describes a tool which has been developed based on this methodology and applied to a complex medical domain where the data to be classified includes temporal relationships. It appears that the method allows for extremely rapid and simple knowledge acquisition and resolves many of the difficulties of dealing with this sort of domain. The present limitations of the method concern making multiple classifications and the probably minor possibility of repetitive knowledge acquisition

---

[†] Present address: The Turing Institute, 36 North Hanover Street, Glasgow G1 2AD, UK.
email: ashwin@turing.ac.uk

* Please send all correspondence to Paul Compton. email:compton@spectrum.cs.unsw.oz.au

Introduction

After more than a decade of research into knowledge-based systems, the problems of knowledge acquisition and maintenance continue to hamper their routine use. While much of this has been attributed to the problems faced by experts in reporting on their mental processes, recent research has taken a different perspective on the nature of the information communicated by experts, and which has repercussions on the knowledge acquisition process (Compton and Jansen, 1990a). This perspective, which views all "knowledge" as merely justifications made in different contexts is shown to provide a framework which may allow experts to construct knowledge-bases without the assistance of a knowledge engineer (Compton and Preston, 1990).

A Justification-Based View of Knowledge

The contention of this justification-based model of knowledge is that the knowledge acquisition process only succeeds in obtaining *justifications* for expert judgement. This is mainly because of the nature of knowledge, rather than the inability of experts to report on their mental processes (Compton and Jansen, 1990a): however this claim is not essential to establish the practical utility of the justification-based model. A feature of these justifications is that they are necessarily dependent on the context in which they are made. For example, for the same case, the justification an expert provides may change depending on whether it is being made to a peer, a layperson, or a knowledge engineer. While theories of non-monotonic and inexact reasoning (Genesereth and Nilsson, 1987) have been developed to capture this contextual nature of the knowledge acquired from experts, they all require an expert to translate the context of their judgement to the requirements of the theory chosen to represent it. As demonstrated by the problems associated with knowledge acquisition, this is particularly difficult (especially as the body of knowledge grows) as it requires maintaining a (consistent) global view of all contexts possible. In contrast to this global view, it is believed here that the nature of the "knowledge" communicated by experts is such that judgement is limited to a context, and new knowledge identifies local changes from one context to another.

This model has formed the basis for developing a "ripple-down" organisation for the rules comprising the knowledge-base (Compton and Preston, 1990) (which is described briefly in a later section). Using this organisation, the authors report that knowledge can be

acquired at least 40 times faster than a conventional approach for a non-trivial expert system (GARVAN ES1) consisting of over six hundred rules (Compton and Jansen, 1990a; Compton and Jansen, 1990b).

Interpreting Chemical Pathology Data

The ripple-down rule redevelopment of GARVAN-ESI (GARVAN RDR) is an expert system for providing clinical interpretations for a specific form of chemical pathology reports, namely, those involving single-date thyroid data. To produce these interpretations, the system uses the ripple-down structure and relatively simple propositions to represent the data. This paper describes a project for developing an expert system to interpret general chemical pathology data. Figure 1 shows one page of such data, as displayed by the reporting software currently used by the Department of Chemical Pathology at St Vincent's Hospital, Sydney. As seen from the figure, the data volume is significant (a patient can have up to a 200 tests, spanning several pages of the type shown) and its interpretation requires the system to reason qualitatively and quantitatively with multi-temporal data: aspects which are common to a range of complex interpretation problems. This paper describes the ripple-down rule structure, and the representation mechanisms that allow an expert to build the knowledge-base for such a task unaided by a knowledge engineer. An important assumption of the domain is that all data available to the expert are also available to the expert system (this is discussed further in a later section). This system is currently being tested and is shortly to be used routinely in an environment reporting on over 500 reports a day.

The Knowledge Acquisition Methodology

Ripple-Down Rules

In the ripple-down structure, rules are organised as a binary tree. The interpreter commences by evaluating the rule at the root node, and proceeds to the right or left nodes depending on whether the antecedent of the rule evaluates to true or false (see Figure 2). In this framework, the conclusion of the expert system is that of the last true rule evaluated by the interpreter. In knowledge acquisition, a case which has been misclassified is run, resulting in a specific path through the tree, and a specific last rule that was tried (the leaf node in that path). A new rule to correctly classify the type of case being considered is then

added to the right or left branch of this last rule tried by the interpreter (depending on its status).

07-Feb-91 ST. VINCENT'S HOSPITAL 10:24 AM
DEPARTMENT OF BIOCHEMISTRY

Provisional Specimen Analysis Report

Patient ..... : CITIZEN,JOHN MA, Male, 54 years

M.R.N. ..... : 00-00-00 Ward : CH 17S
AMO ..... : ADOCTOR

Test	Range	Units	BLOOD 161825 10:42 13 Jul 1990	BLOOD 161928 17:05 13 Jul 1990 VENOUS	BLOOD 161929 17:05 13 Jul 1990	BLOOD 160659 15:11 16 Jul 1990	BLOOD 160884 10:23 17 Jul 1990
Sodium	137-146	mmol/L	136*	....	....	137	139
Potassium	3.5-5.0	mmol/L	3.6	....	....	1.4*	1.9*
Chloride	95-105	mmol/L	91*	....	....	94*	....
Bicarbonate	24-31	mmol/L	38*	....	....	37*	....
Urea	3.0-8.5	mmol/L	8.5	....	....	11.8*	13.9*
Creatinine	0.06-0.12	mmol/L	0.08	....	....	0.07	0.10
Inorg. Phos.	0.70-1.40	mmol/L	....	....	....	0.34*	....
Glucose (F)	3.0-5.8	mmol/L	....	....	....	11.2*	....
Magnesium	0.85-1.05	mmol/L	....	....	....	0.67*	....
Calcium	2.10-2.60	mmol/L	....	....	....	1.84*	....
Albumin	36-47	g/L	23*	....	....	21*	....
Tot. Protein	66-82	g/L	54*	....	....	49*	....
Bilirubin	< 18	umol/L	7	....	....	3	....
Alk. Phos.	30-100	U/L	641*	....	....	698*	....
ALT	< 30	U/L	41*	....	....	51*	....
GGT	< 35	U/L	62*	....	....	60*	....
PCO2	32-45	mmHg	....	48*	42	....	....
PO2	75-105	mmHg	....	36*	69*	....	....
Bicarb.	24-31	mmol/L	....	32*	32*	....	....
Base Excess	-3-+3		....	+8*	+8*	....	....

VENOUS> Venous specimen.

Figure 1: A page of results for a patient

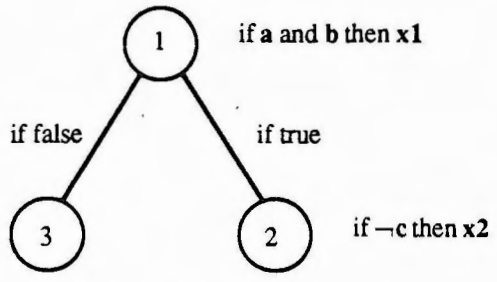


Figure 2: A simple ripple-down tree

Each node in the tree represents the judgement made in the context of the path followed by the interpreter from the root to that node, and the last true rule defines the context that is currently valid. Thus, in Figure 2, **x2** is the judgement made in the context that rule 1 is true but proposition **c** is false. A new rule is added when, given the current data, the expert's view is that the current classification is either incorrect or inadequate. Within this structure, it is considerably easier to acquire rules, since knowledge-base maintenance concerns are minimised to ensuring that the conditions of the new rule are not satisfied by the data that is to be interpreted within the earlier context (in Figure 2, proposition **c** *should* be true for cases intended to have interpretation **x1**). The next section describes a method used by the system described here, that ensures that such errors cannot happen.

Defining the New Context

Although the GARVAN RDR experience indicated a remarkable decrease in the time to construct the knowledge-base, a knowledge engineer was still required to ensure that the new rule added was not satisfied by data associated with the earlier context. Although possible methods of overcoming this were suggested (Compton and Preston, 1990), they remained largely unexplored. This section details the extensions developed that automatically generate the only possible candidates for the conditions of the new rule, leaving the expert to choose one or more of them as changes that define the new context.

Since the primary aim of the knowledge acquisition process is to capture those features of the new context that differentiate it from the earlier one, possible candidates for these features are generated by comparing the current data to that associated with the earlier

context[1]. This includes facts present in the former and absent in the latter, as well as relations that hold in the new context and not in the previous one. For example, if the cases for the two contexts were:

Feature	Old Context	New Context
current Blood_Na	HIGH	LOW
	value=146	value=126
average Blood_K	LOW	LOW
	value=1.4	value=1.9

then, possible candidates that differentiate the new context from the old are:

Feature	Difference
current Blood_Na	value is not HIGH
	value < HIGH
	value is LOW
	value π 146
	value = 126
	x ≤ value, where x Œ [126,146)
	value ≤ y, where y Œ [126,146)
average Blood_K	value π 1.4
	value = 1.9
	x ≤value, where x Œ (1.4,1.9]
	value ≤ y, where y Œ (1.4,1.9]

Assuming the data is complete, at least one of the differences in the second column above must be present when creating a rule that defines the new context. Clearly, it is not necessary for the conditions of the rule to consist only of differences. If the data

---

[1]It should be noted here that the same principles underlie the personal construct approach to knowledge acquisition (Gaines and Shaw 1989), which relies on users identifying important differences between two constructs. It should also be noted that this is consistent with the philosophy of Karl Popper (Popper 1963) who claims that we disprove hypotheses rather than prove them. That is, we show our hypothesis is preferable to the alternatives (in the context) by identifying data which shows (justifies) that our hypothesis is preferable

comprising the cases are represented in set-theoretic terms as **A** (old context) and **B** (new context), then the conditions are such that they contain at least one element from **B - A** and any elements from **A « B** (these may be used by the expert for completeness, or to make a rule more meaningful). Selecting conditions in this way has two major advantages: firstly, it ensures that the rule added does indeed create a new context. Further, the process is sufficiently simple to eliminate the need for a knowledge engineer (at least to manipulate existing concepts).

Reasoning With Time-Course Data

For the time-course data in the domain discussed in this paper, experts use simple functions such as the maximum, average etc., to identify features in the data. A preliminary list of functions currently used by experts enable the determination of:

- The most recent value available for an attribute
- The maximum and minimum values of attribute over the data
- The average value of an attribute over a time period, and the nett change in its values over that period (to obtain rates of change).

The system automatically generates differences in terms of all these functions for each attribute of interest, and the expert merely selects the ones required. It should be noted that while these functions are very simple, they seem to be sufficient for an expert to identify differences between contexts. Experts also sometimes wish to relate attributes to each other (for example, the value of Blood_Ca that occurs at the time of the maximum value of Blood_Alb). There are too many options here to use automatic difference generation. For these special cases the expert interactively specifies such relationships and the differences are again generated automatically. The relationships are again specified using functions (albeit more complex than the ones earlier). Typical relationships used in the interpreting the pathology reports are:

- The value of an attribute at the time instant specified by the current, maximum or minimum value of another attribute (or the nearest available one to this time instant).
- The time difference between a range of possible options. For example, that between the nearest available value of an attribute and the time instant specified by

the current/maximum/minimum values of another attribute. Alternatively, the time difference between the current/maximum/minimum values of one attribute and the current/maximum/minimum values of another.

- Numerical expressions relating the values returned by the functions.

Functions to perform all these tasks are currently encoded into the system. To this extent, there is a role for a programmer when new functions have to be added to the system.

Qualitative Reasoning

A final aspect is the use of qualitative descriptions for the data. In chemical pathology systems, the raw data are predominantly numerical[1]. However, they can often be reduced to broader categories of "high", "normal" etc., by comparing against standard reference ranges. Differences can be generated based on this broader classification (as in the example earlier). Note that it may be possible that although out of the reference range in both cases, the expert could consider data associated with the earlier context clearly high, but that in the new one is normal (due to the circumstances of this context). While such issues cause considerable concern to traditional expert systems, they are easily accommodated in the context-sensitive approach adopted by the ripple-down rule structure. In this, the expert is free to select new cutoffs to (re)define qualitative concepts within the new context (without affecting the definition used earlier: see Figure 3).

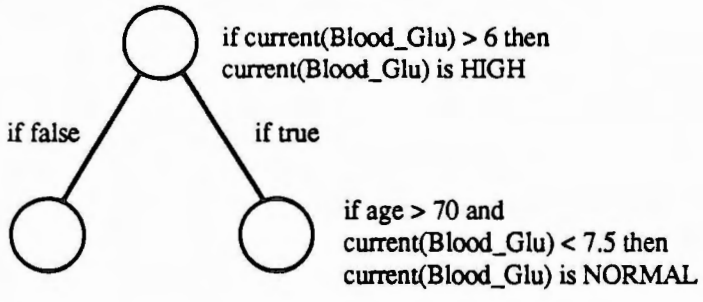


Figure 3: Context-based qualitative reasoning

This makes the decision to accept or reject standard reference ranges clearly context-dependent, which is much more acceptable to a domain expert as it obviates the need to use

[1]Note that the system also allows use of other data as required by experts such as, sex, ward that the referral is from, referring clinician and notes added to the request by the referring clinician. Access to the referring clincian's notes depends on the data entry personnel identifying the type of comment made and entering a code.

global schemes such as fuzzy logic etc. Similarly, experts can easily provide qualitative justifications for their analyses of temporal data using other functions described earlier. To develop general principles for how this data should be handled is extremely difficult (Coiera, 1989) and we are not aware of any conventional expert systems that routinely analyse such data. For the domain considered in this paper, it was found convenient to define two types of qualitative descriptors: those that are ordered and representing states (for example, *very low*, *low*, *high*, *very high* etc.), and those that have no particular ordering and representing qualitative trends (for example, *present*, *increasing* etc.). At any given instant in the time-course data, an attribute can have only one qualitative state, but can exhibit several qualitative trends.

Using this framework, and a medical expert with no previous experience of knowledge based systems, over 100 rules were rapidly added to the knowledge-base in about 3 hours without assistance whatsoever from a knowledge engineer. Note that these are all non-trivial rules that "work", and which will become part of the knowledge-base. A key feature is that the last rule was as easy to add as the first regardless of the complexity of the data (in fact, later rules are probably easier to add as the possible differences between the cases narrows).

Implementation

The existing reporting system for pathology reports at the St. Vincent's Hospital is written in VAX-BASIC and runs under VAX-VMS, producing about 500 reports a day. Software developed for this project includes all aspects of the expert system described above (written in C), as well as programs that interface the system to the existing hospital database. There are two versions of the expert system: an interactive one for use by the expert, and a non-interactive version that runs as a batch process on the cases being reported each day. These two versions are used in the following cycle:

- Prior to each report run the (non-interactive) expert system produces interpretations for all cases about to be reported. The reports are then printed.
- Since it is current laboratory policy that all reports contain a human signature, any errors made by the expert system should be picked up by the expert signing the reports.

- The cases that caused the errors are called up using the interactive version, and the expert forms new rules to correct for these errors.

- Since both versions share the same knowledge-base, no further modifications are necessary prior to the next report run.

The entire process is accomplished with existing personnel in the laboratory and is expected to be in routine use very shortly. It should be fairly easy to control the extra workload this project will generate so that extra staff will not be required. The default for current reporting practice and the expert system is to produce no interpretation. It is therefore possible for the expert to start with a single rule, and add new rules when this rule (and its subsequent refinements) make misinterpretations. As long as the expert does not try to start on too broad a front, we should have a situation where there is gradual but continuous improvement in the quality of interpretations with little impact on the expert's workload.

Discussion

Although the ripple-down rule organisation provides an efficient and simple way of acquiring and maintaining knowledge, there are some important limitations that need to be addressed. The most important one concerns the possibility of repetitions in the knowledge-base. This can happen if the order in which knowledge was acquired was such that:

- Rules testing very specific conditions appear high in the tree. Rules with more general conditions may then be repeated on either side of this tree.

- Because the present structure allows for only a single classification to be made, multiple independent classifications are combined into a single classification. This is acceptable medically but it can result in multiple repetition of the knowledge from one subdomain as additional refinements added to other subdomains

Re-optimising the tree would address the first problem, at the expense of removing the basic feature of the method, that is , storing and using knowledge in context, but it would do nothing to alleviate the second problem of implicit multiple independent conclusions.

We further suggest that in a domain such as chemical pathology the first problem may be minor. For example the ripple down version of GARVAN-ES1 has only 13% of the knowledge repeated. It seems that there is little redundancy in this example (and probably these domains), mainly because data may be complementary rather than independent. Laboratory tests have been developed because of their specificity and sensitivity for patient diagnosis and management and laboratories offer suites of complementary tests in each diagnostic area. It is therefore not very likely that tests A and B will suggest a diagnosis by themselves and test C will suggest an alternative diagnosis regardless of the results of A and B (which is what repetition in the rules implies). Much more likely, and which frequently occurred in the Garvan redevelopment is that a given test result will modify the interpretation of other test results but these modifications will vary depending on the results, which is not repetition.

It seems that the second problem of needing to combine independent areas of knowledge is far more important and solving it may provide a basis for addressing the issue of straight repetition. It is quite likely that a patient with a multi-factorial condition will have a number of independent tests ordered, leading to independent conclusions. At present the ripple down rule approach provides a single interpretation, although that conclusion may contain a number of parts which may lead to large portions of the tree being repeated. The obvious solution of producing separate tree is not attractive because the domains may not be clearly separated. Data from one domain may be used in another and the artificial separation into subdomains may be too crude: resulting in the reoccurrence of the problem within the separate trees. The solution may be to modify the interpreter to backtrack and detect other allowed conclusions in a single tree. This will change the scope of the knowledge acquisition task but not its nature; the expert will still identify differences to develop rules to produce the right interpretations, but will deal with multiple interpretations rather than single interpretations. We anticipate if the system is built like this from the ground up it will not greatly increase the load for the expert. We are currently investigating ripple-down rules with backtracking. It is a significant but not major requirement of a pathology system to handle this type of situation. We propose to minimise the problem at present by judicious choice of initial domains for interpretations. We are dealing with a situation where knowledge acquisition is a negligible effort regardless of the size of the knowledge base. In comparison with conventional knowledge acquisition and maintenance, we judge that the need for some repetitive knowledge acquisition as a minor problem.

Among the other limitations is the fundamental assumption that the expert system has access to all the data available to an expert. While this largely holds, there are some cases in which pathologists recognise patients' names and recall information that is not stored in the hospital's records, and consequently, does not appear in any (machine-generated) case differences. This sort of situation applies to any expert system and the claim must be that the system can only be as expert as a "visiting" expert pathologist rather than the "local" pathologist. However, this is a real and more than satisfactory level of expert performance. A final limitation is that an expert cannot define the actions associated with a new function that he may wish to introduce. While simple functions can be defined using the rule language developed, some programming effort will be required for more complex ones. Whether this can be handled as a knowledge acquisition problem is an area for further research.

Conclusions

We have outlined a method of knowledge acquisition which is based on context and case differences. The method is designed for an environment where there can be an intermediate phase between expert and expert system performing a task, where the expert checks the output of the expert system. We suggest that this method can be applied to very complex classification problems. Initial results suggest that this method provides for extremely rapid and easy knowledge acquisition and is little burden on the expert. A large scale test of this approach is about to commence in a domain where the expert system will be required to make hundreds of different classifications based on over 200 different data variables. It will be expected to produce 500 classifications in batch mode per day.

Acknowledgement

The authors would like to thank Dr. Claude Sammut for his guidance in developing the progams for this project.

References

Coiera, E. "Intelligent patient monitoring". Proceedings of the Fifth Australian Conference on Applications of Expert Systems. Sydney: 1989. 23-43.

Compton, P. and Jansen, R. "Knowledge in context: A strategy for expert system maintenance." Proc AI 88. Ed. CJ Barter and MJ Brooks. Lecture notes in artificial intelligence 406. Berlin: Springer-Verlag, 1990*b*. 292-306.

Compton, PJ. and Jansen, R. "A philosophical basis for knowledge acquisition." Knowledge Acquisition (2) 1990*a*: 241-257.

Compton, P. and Preston, P. "A minimal context based knowledge acquisition system". Knowledge Acquisition: Practical Tools and Techniques. AAAI-90 Workshop. Boston: 1990.

Gaines, B. and Shaw, M. "Cognitive and Logical Foundations of Knowledge Acquisition." AAAI Knowledge Acquisitioh Workshop. Bannf: 1990.

Genesereth, M. and Nilsson, NJ. Logical foundations of artificial intelligence. Los Altos, California: Morgan Kaufmann, 1987.

Popper, K. Conjectures and refutations. London: Routledge and Kegan Paul, 1963.

Techniques and Tools to Support the Knowledge Engineer in Structured Interviewing

Larry E. Wood and John M. Ford

Dept. of Psychology
Brigham Young University
Provo, Utah 84602-1130
USA

Abstract

We have developed an interviewing strategy for knowledge elicitation, which consists of four phases with guidelines for questions suited to each stage. Descriptive Elicitation is intended to reveal important entities and concepts in the domain as reflected in terms and specialized language. Structured Expansion is designed to probe the relationships between the domain concepts and the organization of the expert's knowledge. The Scripting phase relies on the declarative knowledge found through the two previous stages to uncover procedural knowledge, and the final component, Validation, is important throughout the process of knowledge elicitation to ensure that the knowledge being elicited is correct and adequate to solve the class of problems with which it is concerned. We have also developed a prototype "knowledge editor" with which to document and maintain a knowledge base, resulting from interactions with a domain expert. Concepts and their relationships can be displayed as concept maps to view declarative knowledge, and procedural knowledge is represented in structured outlines.

1 Introduction

Recently, Shaw and Woodward [1] proposed a framework in which knowledge acquisition methods and issues can be addressed in a systematic fashion. Their proposals are a reaction to their characterization of the field of knowledge acquisition as a methodological "patchwork", resulting from the importation and adaptation of methods from other disciplines. The concerns expressed by Shaw and Woodward about the field of knowledge acquisition, generally, have been expressed by Forsythe and Buchanan [2] about interviewing techniques for knowledge elicitation, more specifically. They point out that "while recent publications have come closer to addressing some of the

practicalities of interviewing for knowledge acquisition, they still suffer from the general tendency to name methods without providing much information on how to apply them. . . . . In short, we know a good deal about how to represent and manipulate knowledge in expert systems, but have very little systematic understanding of how best to gather that knowledge in the first place." (p. 1-2). They then proceed to report some of their work designed to address this short-coming. Whereas the issues addressed by Forsythe and Buchanan are directed at the social context and communication issues surrounding an interview with a domain expert, the work reported here is concerned with particular kinds of questions asked during an interview and with guidelines for when and how to ask them.

As we have explored strategies for interacting with experts we felt a strong need early on for documentation and editing tools to support and represent the results of those interactions. Thus, we have also developed a prototype of a "knowledge editor," which we have called SNOWMAN (System for kNOWledge MANagement)[1]. Its use in support of our approach to knowledge elicitation will be illustrated throughout the paper.

2 Guiding Principles and Assumptions

Although our approach to interacting with experts is centered around interviewing techniques, we do not mean to imply that we consider interviews to be adequate and sufficient for meeting all the needs of knowledge elicitation. Rather, our goal is to provide guidelines for making interviews with an expert as productive as possible. Also, as will become apparent, our conception of what constitutes interviewing is reasonably broad. Because our approach involves the importation and adaptation of methods from several disciplines, we attempt to provide theoretical bases for them, grounded in those disciplines.

2.1 The Nature of Expertise

As suggested by LaFrance [4], one reason that knowledge elicitation is such a difficult problem for knowledge engineers is because they continue to underestimate the complexity of expertise. Indeed, this is consistent with our own experience and assessment of the knowledge elicitation literature. Thus, one of the guiding principles underlying our approach has been to consider the nature of expertise as it has been characterized by researchers in the discipline of cognitive psychology.

[1] We wish to acknowledge the influence of SemNet [3], an "electronic notebook" system, on the early design phases of SNOWMAN. We express our appreciation to the original developers of SemNet for allowing us access to beta versions.

Four aspects of expertise which we find particularly relevant to knowledge elicitation are: the organization of expert knowledge, the manner in which experts represent problems, experts' characteristic problem solving strategies, and the tacit nature of expert knowledge. We will indicate more specifically how these characteristics have guided our approach as we describe its various phases. For a more detailed treatment of expertise and its implications for knowledge elicitation see [4].

2.2 "Unlocking" Tacit Knowledge.

It is a common observation that much of an expert's problem solving knowledge has become automatized or tacit through extensive use, and there has been considerable debate about its accessibility, even to the expert (e.g., [5] [6]). Anderson's [7] account of the development of automatized knowledge describes this difficulty. According to Anderson, declarative knowledge includes the conceptual entities (both concrete and abstract) in a particular domain and their various relationships. Procedural knowledge, on the other hand, is comprised of the knowledge required for solving problems using relevant domain concepts and relationships. Another way to describe this distinction is a knowledge of "what" vs. a knowledge of "how".

In early stages of skill learning an individual consciously considers various items of declarative knowledge during problem-solving. In well-learned tasks, much declarative knowledge is no longer consciously available during problem-solving, although this does not mean that the declarative knowledge has been "forgotten". It does mean that it may be difficult for an expert to articulate it to a knowledge engineer, especially when asked to do so directly during task performance. While direct access to automatized knowledge may not be possible, an assumption behind our approach is that it is possible to use information about how the expert views the domain to constrain inferences about tacit knowledge. In essence, it involves the systematic use of declarative knowledge to elicit procedural knowledge.

To illustrate this point, we turn to an example from a project to automate the development of bids for a custom cabinet manufacturer[2] with which the second author has been working. The observable data from the bidding task consist of a description of cabinet characteristics, measurements, and individual items such as drawers, turn-shelves, and cabinet tops. Careful investigation reveals that cabinet-maker's expertise includes a classification scheme of types of items which are used for cost determination (e.g., uppers, base, and specialty items). An incautious approach to knowledge elicitation which attempted to relate the nature of items to costs without reference to the abstract

[2] We wish to express our appreciation to Larry Johnson, president of Johnson Custom Cabinets, for his cooperation as the "expert" on this project.

categories used by the expert cabinet-maker would be inadequate in this domain. Thus, accurately capturing this procedural knowledge is dependent on a clear conception of the distinctions inherent in the categorizations used by the expert.

2.3 Simplification Bias

Because a knowledge engineer is often a relative novice in the domain of knowledge being elicited, there exists a potential for errors because of a tendency for the expert to simplify descriptions of domain problem solving. In an attempt to minimize the potential effects of this "simplification bias", we have adapted techniques from the disciplines of anthropology and ethnography because researchers in those disciplines face a similar problem when they are attempting to elicit cultural information from informants (or cultural "experts") in unfamiliar societies. Spradley [8] makes a convincing case for the importance of language in this exercise. He notes that this can be particularly critical when the members of the culture appear to speak the same language as the researcher, because important semantic differences exist that are not readily apparent.

Spradley [8] describes another potential source of bias, which he terms "translation competence". In contrast to simplification bias, which occurs on the part of the knowledge "acquirer," translation competence is a characteristic of the cultural informant. It is expressed when informants translate the reality of their culture in order to explain it to an outsider. The problem is that the more an informant translates for the convenience of an investigator, the more that informant's cultural reality becomes oversimplified and distorted.

In an effort to avoid translation bias, Spradley [8] developed an approach to questioning which makes minimal assumptions (on behalf of the researcher) about informants' knowledge and uses information they provide as the basis for further questioning. The researcher first uses very general probing techniques to persuade the informant to talk freely about his or her domain in a global sense. The informant's language is recorded and examined for category labels and other linguistic cues. This information is then used to probe the informant for additional information. Throughout the process it is necessary to verify that the researcher's emerging understanding of the domain accurately reflects the informant's expertise. We have adopted this general framework in which to develop a knowledge elicitation methodology.

3 Interviewing Strategies

We have developed a four-phase approach to knowledge elicitation which addresses the issues outlined above. As indicated earlier, this approach draws heavily upon

ethnographic methods, but also makes use of techniques from cognitive psychology and the current knowledge elicitation literature. The four phases are: (1) Descriptive Elicitation, which focuses on helping the expert to provide a large quantity of his or her declarative knowledge about the problem-solving domain; (2) Structured Expansion, which has as its focus the expansion and integration of the fragmented portions of the expert's declarative knowledge obtained in Descriptive Elicitation; (3) Scripting, in which the focus changes from obtaining declarative knowledge to using this declarative knowledge to constrain inferences about the expert's "compiled" procedural knowledge; and (4) Validation, which is concerned with the validity of the knowledge engineer's evolving conception of domain problem-solving.

We certainly do not intend the order of presentation to imply that knowledge elicitation can or should be performed in a rigid order. Techniques from all four phases (particularly Validation) may be useful at any time in the elicitation process. However, as suggested earlier, we do rely on making the conceptual structure of declarative knowledge explicit, and using it to guide the elicitation of procedural knowledge (see also [9], [10], [11]).

3.1 Descriptive Elicitation.

In Descriptive Elicitation, the knowledge engineer attends to the categories, objects, models, and other conceptualizations used by experts in describing their domain. Because much of this information is reflected in the expert's use of domain-specific terminology, a significant amount of the knowledge engineer's time in this phase should be spent documenting the expert's use of language. It is important to remember that a word or phrase need not be obviously unfamiliar to the knowledge engineer in order to be important. In the domain of custom cabinet-making, for example, the term 'french drawer joints' has little meaning to a novice and would be an obvious point of focus. What might not be obvious, however, is that the term 'item', has special significance in this domain. The knowledge engineer should attend to all terms and phrases which are used frequently by the expert.

Like many social science investigative techniques , Descriptive Elicitation is based on the use of indirection. The expert concentrates on providing examples and descriptions of different aspects of the domain, while the knowledge engineer attends to the terms and labels which represent domain-specific abstractions. This is an attempt to counteract the possibility that an expert will reflect on automatic processes to such an extent that they will be interrupted or distorted. The goal of Descriptive Elicitation is to direct the expert's attention toward the task of describing the structure of the problem-solving domain. It is important for the knowledge engineer to attempt to create a situation in which the expert

will describe the the domain in a natural way, using domain-specific terms for important concepts. However, the knowledge engineer must also be concerned about keeping the expert focussed on that portion of the domain which has been targeted for knowledge elicitation. Otherwise, the interview can become very inefficient. Specific types of questions are designed to assist the knowledge engineer to meet these various goals.
One way to elicit a large sample of expert terminology is to ask a number of Grand Tour questions which encourage the expert to verbally "show the knowledge engineer around" the physical, temporal, and abstract space of the problem-solving domain. Examples of such questions are shown in Table 1. Grand Tour questioning is of particular utility early in the elicitation process, but may be used whenever the expert identifies a new subproblem. For knowledge elicitation, we have found one of the most useful types of questions to be Task-Related and Guided Grand Tour questions (see Table 1). Because the Knowledge Engineer's ultimate goal is to elicit the knowledge required to perform some class of problems (tasks), these types of questions help keep the expert focussed on relevant aspects of domain knowledge, without causing too narrow a focus too early. Asking about typical problem-solving episodes often elicits general categories used in frequently occurring situations. Some experts, however, may be uncomfortable with this degree of abstraction. Such situations call for the use of Specific Grand Tour questions (see Table 1) which refer to more concrete situations.

Case-Focused questions (see Table 1) are also useful in eliciting domain terminology. Cases are indexed in memory by the categories of which they are members. These types of questions elicit further terminology used to characterize domain problem-solving by evoking the expert's description of the characteristics and details of particular problem-solving episodes. They also provide a basis for keeping an interview focussed on relevant topics, especially if an expert tends to get side-tracked.

Example questions (see Table 1) help the knowledge engineer to obtain more detail about terms already identified. Personal Experience questions (see Table 1) are also useful for this purpose, but can be a source of distraction if used too early in the elicitation process. The most useful domain abstractions for early knowledge engineering efforts are those which are relevant to many potential cases in a domain. Second-Hand Experience questions (see Table 1) also suffer from the atypicality problem, but provide the knowledge engineer with indirect access to the larger "community" of domain experts. Communication between experts within this community about domain problems and knowledge is one of the processes which fosters the development and use of specialized domain terminology [8]. If two or more experts are readily available for knowledge engineering, it is useful to observe them as they discuss various cases e.g., [12].

Although Descriptive Questioning is designed to use indirection to elicit natural domain description, there will undoubtedly be times when this strategy will fail. As

discussed earlier, Spradley [8] warns that cultural experts tend to translate their knowledge into terms they believe the interviewer will find easier to understand. Native Language questions (see Table 1) can be used when this is suspected. Although it is difficult to tell exactly when an expert is translating, there are some cues.

Table 1 Descriptive Questions

Question Type	Example
Grand Tour	
Task-Related	"Could you discuss the steps you go through in generating a bid?"
Guided	"Could you show me a 'take-off sheet' and point out the different parts?"
Typical	"Could you tell me about a typical visit with a potential customer?"
Specific	"Tell me about the last visit you had with a potential customer, from start to finish."
Minitour	"Tell me about a typical customer request for a new counter top?"
Case-Focused	
Example	"Can you give me an example of estimating the cost of an 'appliance center'?"
Personal Experience	"You've probably had some interesting experiences dealing with requests for 'special molding'. Tell me about some of them."
Second-Hand Experience	"What kinds of experiences have you heard other cabinet makers tell each other about determining costs when there are requests for a 'unique stain'?"
Native-Language	
Direct Language	"What do you call shelves that are round and can be swiveled?"
Hypothetical-Interaction	"How would you describe cost estimation for 'pop-up shelves' to another cabinet maker?"
Typical Utterance	"What are some sentences or phrases I might hear cabinet makers use that contain the term: 'item'?"

One technique often used by ethnographers [8] to elicit informants' natural descriptions of terms and concepts is to ask how terms, tools and concepts are "used"

rather than what they "are" or "mean." Direct questions about meaning tend to encourage the expert to translate. Queries about use prompt the informant to describe the context in which a particular term or object plays a role. This description provides information which can be used to structure more in-depth questioning. A use-oriented question from the cabinet-making domain is: "What purpose does a 'splash' serve?"

Following an interview, our knowledge editor (SNOWMAN) can be used to record information in forms that can be reviewed and revised. SNOWMAN is a prototype knowledge editor implemented in SuperCard™. It is assumed that a knowledge engineer takes notes, audio recordings, video recordings, or some combination of the above to preserve a record of the elicitation interviews. Following the interview, these records are reviewed and the expert's terminology describing objects and relationships can be entered into SNOWMAN. To illustrate this process, we will analyze some of the information provided when the following Task-Oriented Grand Tour question was asked of a master cabinet maker: "Describe the steps you go through to generate a bid for a customer." Part of the answer to the question was, "I begin by asking customers for their preferences for things such as style, type of wood, stain, and finish."

Declarative knowledge is entered into SNOWMAN by individual concepts or objects and their relationships to other concepts and objects. One way to interpret information provided by the cabinet-maker is that cabinets have certain features for which customers must choose values in specifying their custom designs. Thus, the concept "cabinet" could be entered into SNOWMAN by indicating it has "styles", and features such as "wood", "stain", and "finish." The concepts are entered via the dialog box shown in Figure 1. Thus, each concept and its relationships to other concepts is stored along with any explanatory information and/or synonyms.

The information about a particular concept can be displayed as shown in Figure 2. Any of the concepts to which a displayed concept is related may also be displayed by clicking the mouse on its name in the list of related concepts. Thus, it is possible to "navigate" through the concept structure of the knowledge base as desired. It is also possible to create a concept map, displaying the network of relationships a particular concept has with others in the knowledge base. A small portion of a concept map for "cabinet" is shown in Figure 3. The names of all concepts and of all relationships contained in a particular knowledge base are stored in alphabetized lists. The concepts list is used to select individual concepts for inspection, and the relationships list is used to select relationships that have already been defined. The information from each interview with the expert is used to extend and revise the knowledge base.

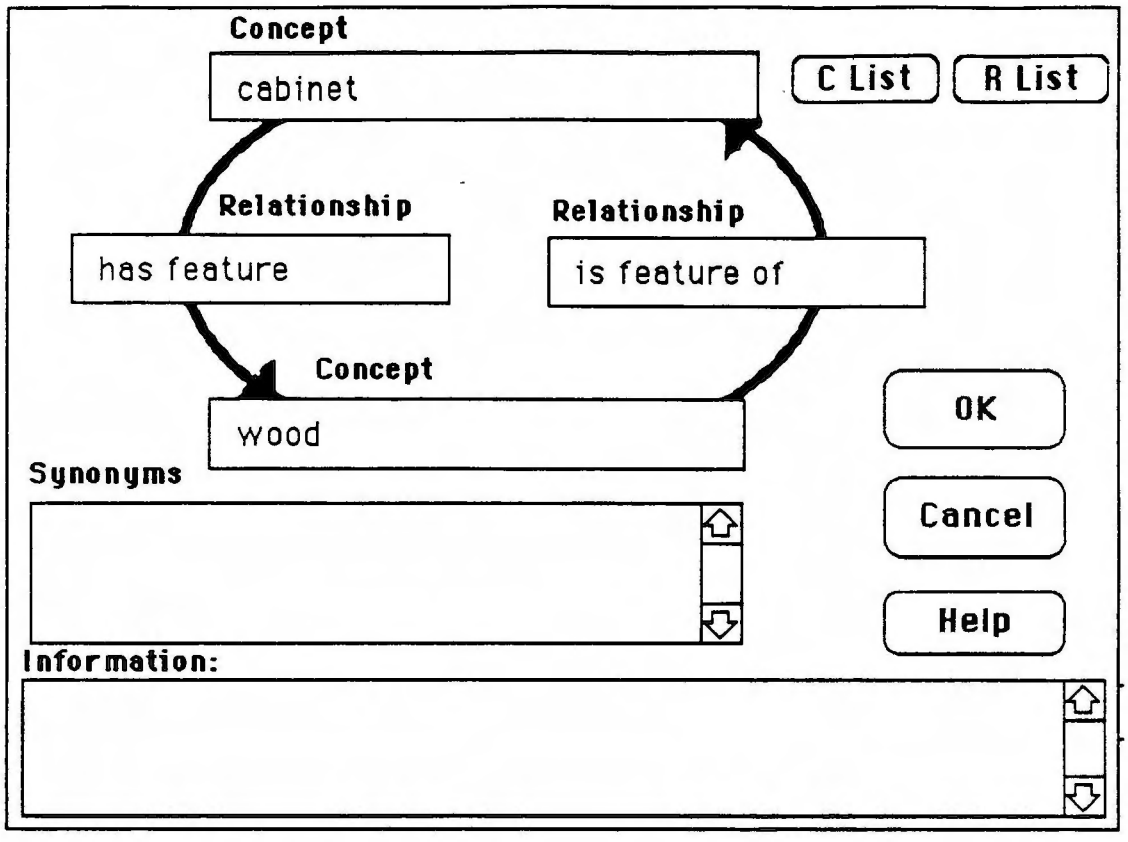


Figure 1. Dialog box for entering concepts.

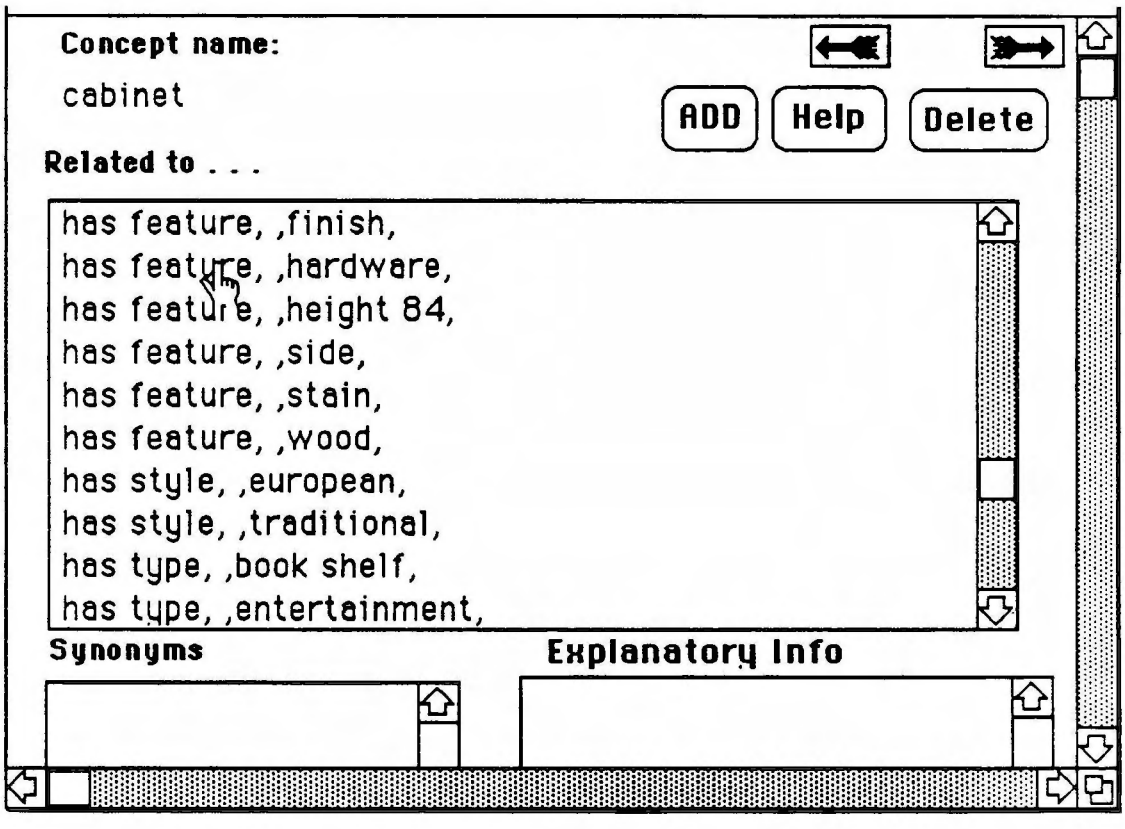


Figure 2. Display of information concerning a single concept.

3.2 Structured Expansion.

In Structured Expansion the knowledge engineer uses questioning techniques that explore the rich, integrated organizational structure of the expert's knowledge. These techniques are based on the general principle that concepts in memory are associated with one another, that remembering or being cued with one or more domain concepts will result in the recall of additional relevant concepts. It is helpful here to keep in mind that studies (e.g., [13]) have shown that at a macro level expert knowledge is generally organized hierarchically (i.e., categories with sub-categories) although many other types of relationships are also present, depending on the particular domain. At the micro level, expert knowledge is stored as organized "chunks" of frequently occurring patterns or schemas e.g., [4].

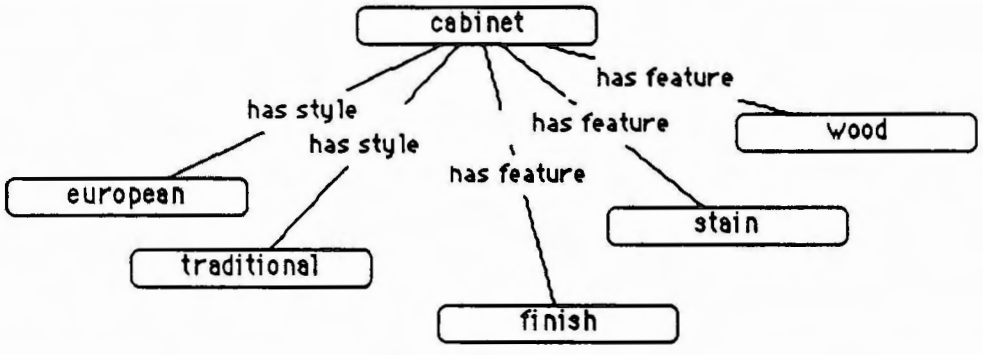


Figure 3. A portion of a concept map display from an initial elicitation session.

The first technique, and perhaps the most important, given our focus on language, is to phrase questions to the expert in domain-specific terminology. A knowledge engineer's consistent use of terminology different from that of the expert, will encourage the expert to translate. Using the domain-specific language, however, provides a context of familiarity and encourages the expert to focus on the domain itself rather than the knowledge engineer's methods and unfamiliarity with the domain.

There are several additional ways to cue experts to provide information while minimizing bias. One technique used by linguists [14] is to simply increase the length of the questions. Experience has shown that when even the simplest question is accompanied by an explanation, an expression of appreciation, or any non-leading discussion, then the interviewee will provide a longer response. In addition to being longer, the response is less likely to be distorted. When an expert answers a brief direct question, responses are made to those associations in memory evoked by the cues in the question. On the other hand, when an expert provides a prolonged response or description, there is less direct influence from the interviewer.

When possible, it is preferable to elicit knowledge in the setting in which the expert usually solves problems [15]. The familiar surroundings serve as further cues to the knowledge used by the expert in domain problem solving. The questioning context also has an effect on the amount of information provided by the expert. Similar questions should be asked in several contexts to elicit complete information. Also, there is evidence [16], [17] indicating that individuals select a limited number of all possible associations for report in any given interviewing context. It would be premature to conclude that knowledge about a given term or topic is "tacit" and completely unavailable to an expert when he or she fails to respond in a single questioning context. Rather, it is advisable to extend questioning about each term over a number of interviewing contexts in order to maximize the total amount of information retrieved.

These considerations are important, but not sufficient alone to guide further elicitation efforts. Unstructured free association by itself is not a productive method for building models of expert knowledge. Additional information about the form or content of the expert's knowledge is necessary if questioning is to produce a coherent and organized model of that knowledge. Ethnographers have developed methods for eliciting hierarchies of concepts based on language cues. Examples of questions based on these principles are presented in Table 2. The knowledge engineer needs to look for "cover terms" or terms denoting what we might call concepts. A particular cover term may associated with a number of "included terms", each of which are related to the cover term by a particular semantic relationship. All of the included terms related to a given domain concept via the same relationship constitute a "contrast set" among which meaningful distinctions are made.

Suppose, for example, the expert said "There are different kinds of wood, such as oak and walnut." This indicates that there are at least two included terms in the domain defined by the cover term 'wood' and the semantic relationship **kinds of**. These two entities are significantly alike in being **kinds of** wood and it is likely that the expert can provide further information about other kinds of wood and make meaningful distinctions among them. The most fruitful way to go about the task of eliciting such domain structure is to first identify the cover term and semantic relationship from a sample of expert language elicited during descriptive elicitation activities. Then the knowledge engineer should elicit as many included terms as possible by asking Cover Term questions (see Table 2) or by having informants fill in Substitution Frame questions (see Table 2) in as many ways as possible. Once the informant indicates that there are no more included terms in a particular domain, the knowledge engineer should elicit the relevant features and dimensions of contrast which discriminate the included terms from one another. This is accomplished by asking a number of Contrast questions (see Table 2). As before, it is important to attend to the experts terminology for these dimensions and features.

Table 2 Structural Questions

Question Type	Example
Relationship	
Cover Term	"What are the different kinds of 'wood' that are available for cabinets?"
Included Term	"Are 'raised' and 'recessed' both types of 'arched wood doors'?"
Substitution Frame	"How many different ways can you think of to complete the sentence 'Customers usually don't understand _____ is available'?"
Contrast	
Directed Contrast	"Could you look through this 'take-off sheet' and show me what other items are considered 'specialty items'?"
Dyadic Contrast	"What differences are there between 'end panels' and 'finish panels'?"
Triadic Contrast	"Which two of the three cabinet types listed on this page are alike and which one is different from the other two?"
Sorting	"Please sort these cards [cabinet features] into two or more piles, depending on how they are alike or different."
Rating/Ranking	"What types of wood do you like best and which are your least favorite?"

The best place to begin building domain hierarchies may be with "basic-level" domain terms [18], [19], [20] which can be tentatively identified by their frequency of use by the expert, their lexical brevity (note the prevalence of acronyms and other abbreviated forms in the jargon of many domains), and, to a lesser extent, by their unfamiliarity to the knowledge engineer. The use of Contrast questions (see Table 2) for more information at this level exploits the larger number of distinctions and associations to these key concepts in the expert's memory.

Boose and Bradshaw [21] state that the most profitable unit of analysis for knowledge base development is the "distinction", but they admit that attempting to elicit distinctions without some form of constraining context can be problematic. Confusion may result when experts are requested to create distinctions between elements at what the authors describe as different "levels of abstraction." Thus, if a knowledge engineer fails to give proper concern to domain structure, artificial distinctions may be elicited from the expert which are not relevant to problem solving.

Limiting requests for distinctions to contrast sets of included terms helps ethnographers avoid the problem of eliciting artificial distinctions. They have found that informants will give "test question responses" in an attempt to provide the "right" answer and will make artificial distinctions, resulting in an inaccurate portrayal of their knowledge. Our focus on the structure of expert domain knowledge is an attempt to avoid this difficulty. It should be noted here that although we have emphasized categorical relationships in our discussion, there will obviously be other important relationships that must be captured (e.g., part-whole, attributes, cause-effect), depending on the domain under study [22]. A sample of extensions to the knowledge base in the cabinet-bidding domain produced through Structured Expansion is shown in Figure 4.

3.3 Scripting.

Once the content and structure of a large portion of the expert's declarative knowledge has been elicited, this information can be used to guide the search for procedural expertise, or the ways in which expert use the concepts and objects of declarative knowledge to solve domain-specific problems. We refer to this phase of elicitation as "Scripting". It is useful at this stage to bear in mind two of the characteristics of expertise. First, it has been shown (e.g., [13]) that experts represent problems at a level of underlying principles and abstractions. Second, the strategies which experts use in solving problems are sometimes common across domains [23] and sometimes domain specific [24]. The particular representations and strategies are often elusive to the knowledge engineer and must be explicitly sought by a variety of methods, as we discuss below.

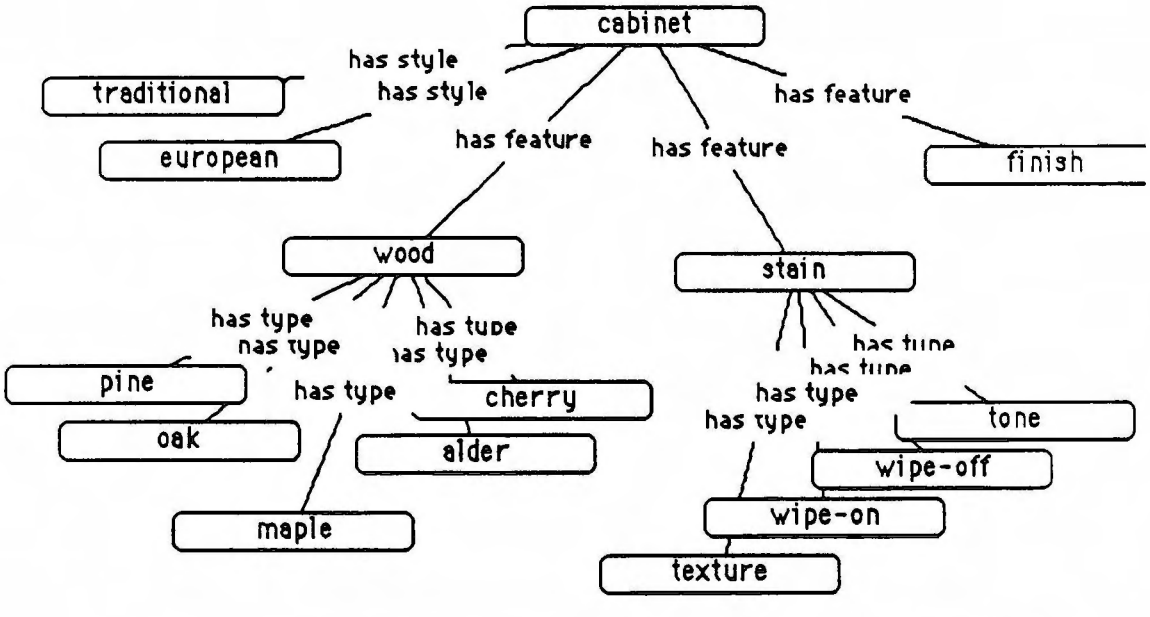


Figure 4. Expanded concept map resulting from Structural questioning

Perhaps the most widely-used technique developed in cognitive science to investigate problem solving strategies is think-aloud protocol analysis. The term "protocol analysis" represents a family of techniques used to examine subject's verbalizations about their problem-solving processes. Ericcson and Simon [5] have observed that the analysis of a verbal protocol must be preceded by a domain definition phase, during which the researchers must exhaustively catalogue the labels for objects, concepts, etc. referred to in the protocol. The purpose of this phase is to provide researchers with the declarative knowledge necessary to interpret subjects' statements about the processes involved in problem solving.

The utility of using declarative knowledge to interpret verbal protocols is the reason we recommend obtaining extensive declarative knowledge before attempting to elicit detailed procedural knowledge. Once the important concepts and objects in the domain are understood by the knowledge engineer, a think-aloud protocol is much easier to interpret, and the effort required for analysis is minimized. When emphasizing elicitation methods as we are doing here, methods for protocol generation simply become additional interviewing techniques, although they are usually not categorized with interviewing techniques in the knowledge acquisition literature e.g., [1].

Reasoning strategies emerge from a variety of think-aloud protocols generated by asking experts to describe their thought processes as they solve specific problems or cases. One of the important aspects of expertise to be captured is the set problem solving strategies used by the expert. As more and more knowledge is obtained, it is used to guide and constrain subsequent elicitation efforts.

There are at least three types of protocol-generating procedures that are useful in knowledge elicitation. As shown in Table 3, Think Aloud questions simply request an expert to report the contents of conscious awareness during the

Table 3 Scripting Questions

Question Type	Example
Think Aloud	"Think out loud while you generate a bid for the kitchen from this take-off sheet. Do not worry about talking in complete sentences or 'making sense'."
Aided Recall	"As we listen to this tape of the bid you just generated, say whatever you remember about what you were thinking. Stop the tape or rewind whenever you want."
Cross-examination	"Where did you account for the cost of the special finish on this section of 'uppers'?"

solution of a domain problem. Ericsson and Simon [5] caution that this procedure can only yield meaningful results when the task being performed concurrently with the protocol does not require the use of verbalization or other processes which tax the resources of working memory. Even when this condition holds, a subject is likely to provide an incomplete account of the moment-to-moment contents of consciousness. Information which is processed automatically will not be reported at all. However, these results may still be valuable because they provide information about the sequence in which the expert attends to various problem features and the various subgoals for which these features are relevant.

Aided Recall questions (see Table 3) can be used to follow-up results obtained in think-aloud protocols. This technique, which has been used successfully in eliciting descriptions of expertise [25], [26], involves video or audio-taping the performance of an expert engaged in domain problem-solving and then reviewing this record with the expert at a later time. This procedure frees the expert from the interference resulting from think-aloud protocols and creates a situation in which recall of processing is cued by an unbiased record of the expert's own performance. A variation on aided-recall , called cross-examination (see Table 3), has been used successfully by Kuipers, Moskowitz and Kassirer [27] to probe the limits of knowledge, directly. After completing a think-aloud protocol on a particular problem, the expert is asked specific questions about it.

These methods can be supplemented by the knowledge engineer's use of the previously elicited declarative knowledge to interpret and to fill in gaps in the protocols. When an expert mentions a particular concept or term, the knowledge engineer can use knowledge of the expert's conceptual framework to infer other relevant concepts which may be playing a role in decision-making at that moment. The knowledge engineer is still required to do some speculating about the expert's reasoning processes, but these speculations will now be constrained by several different types of information.

In SNOWMAN, we currently represent procedural knowledge in the form of structured outlines as prescribed by Merrill [28] for task analysis. Structured outlines are a part of structured analysis, wherein primitive control structures are used to represent the structure of a task. The primitive control structures (adapted from structured programming) are: (1) linear sequences or steps, (2) alternate paths resulting from decision points, and (3) repetitive actions performed until some criterion is reached. The resulting analysis is represented in a structured outline, using vocabulary and syntax of natural language to represent the primitive sequence structures and using indentation to show more detailed representations of global structures.

Structured outlines have several advantages. First, they provide a means of representing knowledge at the global, strategic level, while at the same time representing more specific knowledge within a particular structure. Second, the structure of the outline

can easily be modified to reflect changes in understanding that emerge from successive sessions with an expert. Finally, the outlines can be mapped to graphical representations such as flowcharts, decision tables, or decision trees, if desired.

We have found it convenient to enter segments of procedural information as it appears during early phases of knowledge elicitation, and then fill out details through analysis of think-aloud protocols generated in the context of specific problems, cases, or examples. For example, the following general outline of major steps in the generation of bids was obtained from the first elicitation session with our expert cabinet-maker:

```
DETERMINE Job Type
ESTABLISH costs for bid
CALCULATE costs
PREPARE  Bid Specifications document
```

Following the scripting phase, the major steps of the outline were further refined and expanded as follows:

```
DETERMINE Job Type
    IF  type of job is remodeling
       THEN  provide a time and material bid for preparation
END IF

ESTABLISH costs for bid
    Establish information applicable to entire job
        Cabinet Style
            SELECT CASE
                IF  traditional, use default = $75/ft
                IF  European, reduce the price/ft. costs by 15%
            END SELECT
        Drawer Joints
            SELECT CASE
                IF  french  use default = $75/ft
                IF  rabbit subtract $3/drawer
                IF  finger add $8/drawer
            END SELECT
        •
        •
        •
ESTABLISH costs for bid  (cont'd)
    Establish information applicable to an entire room.
        Wood Type
            SELECT CASE
                IF  oak, use default = $75/ft.
                IF   maple, use default = $75/ft.
                IF   pine, subtract $4/ft.
                IF  walnut,  add $7/ft.
                IF  alder, subtract $6/ft.
                IF  cherry, add $7/ft.
                IF  willow, subtract $4/ft.
        END SELECT
```

```
    .
    .
    .

CALCULATE costs
    REPEAT FOR EACH room
        REPEAT FOR EACH item in a room
                cost = #items x cost/item
        END REPEAT
    END REPEAT
    Calculate total cost of all items in bid.

PREPARE  Bid Specifications document
```

Presently, Structured Outlines are implemented in SNOWMAN simply by having an editable text field into which outlined text can be entered. There are menu items allowing "empty" structures to be automatically entered with appropriate tabbing, which the knowledge engineer can then fill in, where that is convenient. Examples of empty If-Then-Otherwise and Select-Case structures are as follows:

```
IF condition
    THEN operations
    OTHERWISE operations
END IF

SELECT CASE
    IF condition1  operation1
    IF condition2  operation2
    IF condition3  operation3
END SELECT
```

While the representations used in SNOWMAN are designed to be of benefit to the knowledge engineer in building and maintaining the knowledge base, one of their benefits has been as useful communication aids with the expert in subsequent interviews. They can provide a significant portion of the agenda for an elicitation session by having the expert systematically review them, recommending additions and changes. In fact, one of our current research efforts is directed toward additional representations both for the benefit of the knowledge engineer directly and for more effective communication with experts.

3.4 Validation.

It should be obvious that a constant concern during knowledge elicitation activities is to capture an accurate view of the domain and the procedures used by the expert to solve the class of problems under consideration. We divide validation activities into "checks," or attempts to validate knowledge in the possession of the knowledge engineer, and "controls," or methods designed to reduce error and misunderstanding in the knowledge

elicitation process. All of the elicitation methods we have described above, however, can be classified as controls We believe that following a careful and controlled methodology which minimizes bias can be at least as important as rigorous post-elicitation checking. It is important to include both checks and controls throughout the knowledge elicitation process. The following general principles characterize effective use of checks.

Checks should be conducted during all stages of knowledge elicitation. The list of questions in Table 4 contains sample check questions which apply to knowledge from each of the other three phases of elicitation. These and other similar questions should be used to verify every type of knowledge gathered during elicitation. Case Verification questions (see table 4) in which the knowledge engineer attempts to solve domain problems and receives feedback from the expert plays a very important part in validation. Asking the expert to review printed copies of the of representations from SNOWMAN has been very beneficial in promoting ongoing validation of the knowledge base.

Evidence from cognitive psychology [29] as well as from the naturalistic inquiry tradition [30] suggests that people are prone to a systematic "verification bias" when validating knowledge. For example, in validating information such as "women are easier to interview on the telephone than men" there is a tendency to seek and remember confirming evidence (e.g. hard-to-interview men and easy-to interview women) and to ignore disconfirming evidence (e.g. easy-to-interview men or hard-to-interview women).

Table 4 Validation Questions

Question Type	Example
Native Language	"Would you call this a 'take-off sheet' when talking to another cabinet-maker?"
Domain	"Are there different kinds of 'items'?"
Included Term	"Is a 'pop-up shelf' a kind of 'base' cabinet?"
Relationship	"Is it true that 'European' is a style of cabinet?"
Contrast	"Would it be accurate to say that 'finish' panels are in the middle of a cabinet section while 'end' panels close off a cabinet section?"
Sequencing	"The steps in calculating costs for a section are: 1) determine type of cabinet; 2) record the measurement; 3) consider job and room characteristics that affect cost; and 4) multiply number of items and cost/item. Is that right?"
Case	"Let's go through a room and I'll play the role of a cabinet-maker using the knowledge base to select the appropriate action at each step. Tell me when I do something that you would not."

Ethnographers adopt two strategies to counter this tendency which we find useful. First, they make the analysis of these "negative cases" an explicit goal of elicitation. The second, more general strategy is to ask an outsider who is neither a domain expert nor directly involved in the elicitation process to audit or review the elicitor's notes on the domain. Experience has shown that this process can uncover "blind spots" such as those produced by verification bias.

When errors are discovered, the knowledge engineer should attempt to determine the cause of the error. Discovering that knowledge is incomplete or only applicable in special contexts is not a cause for concern, but does mean that further elicitation efforts are necessary. If, on the other hand, the expert claims that the knowledge is "wrong," then the knowledge engineer should examine the possibility that this error was produced by faulty or incautious knowledge elicitation. In particular, the knowledge engineer should seek to determine whether she or he has been systematically oversimplifying domain knowledge.

4 Summary & Conclusions

We have described a methodology for structuring interviews with domain experts during knowledge elicitation. We have developed our approach by importing and adapting methods from disciplines which have a history of related activities. In doing so, we have been influenced by four major considerations. First, we have been guided by the cognitive nature of expertise, some important aspects of which are: (1) a high degree of tightly integrated domain-specific knowledge, (2) problem representations or schema which are based on underlying domain principles and abstractions, (3) general and domain-specific problem solving strategies present in a particular domain, and (4) the fact that a substantial portion of the knowledge is tacit and difficult to articulate.

A second consideration which has influenced our efforts is a concern with having a knowledge engineer avoid simplification bias. This has led us to place a heavy emphasis on the use of domain-specific language and interviewing methods that attempt to prevent the experts from exercising their translation competence.

A third consideration guiding our development efforts has been the early elicitation of declarative knowledge which is then used to guide the elicitation of procedural knowledge. This is one attempt to address the difficult problem of tacit knowledge. The final consideration influencing our efforts is a need for a knowledge editor that allows knowledge to be recorded and maintained while developing a knowledge base and to assist the knowledge engineer to communicate its contents to the expert. This latter consideration reflects our preference for creating as detailed a model of the expert's

problem solving as possible before getting caught up in the details of system implementation.

Given the aforementioned considerations, we have developed a four-phase approach with guidelines for questions suited to each stage. First, the Descriptive Elicitation phase is intended to reveal the important entities and concepts in the domain as reflected in the terms and specialized language used by the expert. A second phase, Structured Expansion, is designed to probe the relationships between the domain concepts and the organization of the expert's knowledge using the terminology uncovered in the previous stage. The third phase, Scripting, relies on the declarative knowledge found through the two previous phases to elicit procedural knowledge, at both macro and micro levels. A final phase, validation, is important throughout the process of knowledge elicitation to ensure that the knowledge being elicited is correct and adequate to enable a system in which it is implemented to solve the class problems with it is concerned.

In developing our methodology, we have drawn upon the methods of a number of disciplines, particularly in the social sciences. Other knowledge elicitation theorists such as Brulé and Blount [31], Diaper [32], Forsythe and Buchanan [2], and LaFrance [33] have recommended similar approaches. We are convinced the field of knowledge elicitation will continue to benefit from this type of interdisciplinary effort. Furthermore, we are designing a series of empirical experiments to determine the effectiveness of our approach compared to less structured approaches or approaches which use alternative types of structure. We believe that such studies are a necessary complement to formative evaluation that comes from the use of the techniques in knowledge elicitation projects.

We have also found a number of additional capabilities that would enhance the use of SNOWMAN. One feature that is lacking is a way to focus on portions of a large semantic network or structured outline without losing the context of which it is a part. We think the notion of fisheye views [34] has considerable promise in that regard, and we intend to experiment with them. In addition, while the structured outlines are very useful for entering and revising procedural representations, we are also investigating more graphically-oriented representations such as flow-charts, logic diagrams, and decision tables. Also, there are no close computational links between the declarative and procedural representations, and we are exploring ways to rectify that. Finally, because the representations currently implemented in SNOWMAN have proven useful not only for knowledge engineers but also in communicating with experts for purposes of validation, we are actively researching methods and representations that can be used specifically for the latter purpose.

References

[1] Shaw MLG, Woodward, JB. Modeling expert knowledge. *Knowledge Acquis* 1990; 2:179-206.

[2] Forsythe DE, Buchanan BG. Knowledge acquisition for expert systems: Some pitfalls and suggestions. *IEEE Trans on Systems, Man, and Cybern* 1989; 19:435-442.

[3] Fisher KM, Faletti J, Patterson H, Thornton R, Lipson J, Spring C. Computer-based concept mapping. *Journal of College Science Teaching* 1990; IX:347-352.

[4] LaFrance M. The quality of expertise: Implications of expert-novice differences for knowledge acquisition. *SIGART Newsletter* 1989; 108 (April):6-14.

[5] Ericsson KA, Simon HA. *Protocol analysis: Verbal reports as data.* MIT Press, Cambridge, Massachusetts, 1984

[6] Nisbitt RE, Wilson, TD. Telling more than we can know: Verbal reports on mental processes. *Psychol Review*, 1977; 84:231-259.

[7] Anderson JR. *The architecture of cognition.* Harvard University Press, Cambridge, Massachusetts, 1983

[8] Spradley JP. *The ethnographic interview.* Holt, Rinehart and Winston, New York, 1979

[9] de Greef P, Breuker J. A case study in structured knowledge acquisition. *In Proceed of the Ninth Internat Joint Confer on Artif Intell.* Morgan Kauffmann Publishers, Inc., Los Altos, California, 1985

[10 Cooke NJ. The elicitation of domain-related ideas: Stage one of the knowledge acquisition process. In: Ellis C (ed) *Expert knowledge and explanation: The knowledge-language interface.* (pp. 58-75). Wiley & Sons, New York, 1989

[11] Roth, EM, Woods DD. Cognitive task analysis: An approach to knowledge acquisition for intelligent system design. In: Guida G, Tasso C (eds) *Topics in expert system design: Methodologies and tools.* (pp. 233-264). North-Holland, Amsterdam, 1989

[12] Cochran EL, Bloom CP, Bullemer P. Increasing the end-user acceptance of expert systems by using multiple experts: Case studies in knowledge acquisition. In McGraw KL, Westphal CR. (eds) *Readings in knowledge acquisition: Current practices and trends*. Ellis Horwood, LTD, London, 1990

[13] Chi MTH, Feltovich, PJ, Glaser R. Categorization and representation of physics problems by experts and novices. *Cognit Science* 1981; 5:121-152

[14] Churchill L. *Questioning strategies in sociolinguistics*. Newbury House Publishers, Rowley, Massachusetts , 1978

[15] Bell J, Hardiman RJ. The third role -- the naturalistic knowledge engineer. In Diaper D (ed) *Knowledge elicitation: Principles, techniques, and applications*. (pp. 49-85). John Wiley & Sons, New York, 1989

[16] Mitchell AA, Chi MTH. Measuring knowledge within a domain. In Nagy P (ed) *The representation of cognitive structures*. Ontario Institute for Studies in Education, Toronto, 1984

[17] Werner O,Schoepfle GM. (eds) *Systematic Fieldwork: Vol. 1. Foundations of ethnography and interviewing*. Sage Publications, Newbury Park, California, 1987

[18] Brown R. How shall a thing be called? *Psychol Review* 1958; 65:14-21

[19] Lakoff G. *Women, fire, and dangerous things: What categories reveal about the mind*. University of Chicago Press, Chicago, 1987

[20] Rosch E. Natural categories. *Cognit Psychol* 1973; 4:328-50

[21] Boose J, Bradshaw J. Expertise transfer and complex problems: using AQUINAS as a knowledge acquisition workbench for knowledge-based systems. *Internat Journal of Man-Machine* Studies 1987; 26:3-28

[22] Merrill MD. Content and instructional analysis for cognitive transfer tasks. *AV Commun Review* 1973; 21(1):109-125

[23] Anderson, JR. *Cognitive Psychology and Its Implications*. Freeman, San Francisco, 1990

[24] Anderson JR, Farrell R, Sauers, R. Learning to program in LISP. *Cognit Science* 1984; 8:87-129

[25] Grover MD. A pragmatic knowledge engineering methodology. In *Proceedings of the American Association of Artificial Intelligence*. Morgan Kauffmann Publishers, Inc., Los Altos, California,1983

[26] Kuipers BJ, Kassirer JP. How to discover a knowledge representation for causal reasoning by studying an expert physician. In *Proceedings of the American Association of Artificial Intelligence*. Morgan Kauffmann Publishers, Inc., Los Altos, California, 1983

[27] Kuipers B, Moskowitz AJ, Kassirer JP. Critical decisions under uncertainty: Representation and structure. *Cognit Science* 1988; 12:177-210.

[28] Merrill PF. Job and task analysis. In Gagne' RM (ed) *Instructional technology: Foundations*. (pp. 141-173). Lawrence Erlbaum & Associates, Hillsdale, New Jersey, 1987

[29] Johnson-Laird PN. *Mental models: Towards a cognitive science of language, inference, and consciousness*. Harvard University Press, Cambridge, Massachusetts, 1983

[30] Lincoln YS, Guba EG. *Naturalistic Inquiry*. Sage Publications, Beverly Hills, California, 1985

[31] Brulé JF, Blount A. *Knowledge acquisition*. McGraw-Hill Publishing Company, New York, 1989

[32] Diaper D (ed). *Knowledge elicitation: Principles, techniques, and applications*. John Wiley & Sons, New York, 1989

[33] LaFrance M. The knowledge acquisition grid: a method for training knowledge engineers. *Internat Journal of Man-Machine Studies* 1987; 26:245-255

[34] Furnas GW. Generalized fisheye views. *Proceedings of the ACM SIGCHI Conference*. ACM Press, New York, 1986

Fifth European Knowledge Acquisition for Knowledge-Based Systems Workshop

EKAW91

Generic Task Methodologies

Clamping, COKAM, KADS, and OMOS:
The Construction and Operationalisation of a KADS Conceptual Model

Otto Kühn*, Marc Linster+ and Gabriele Schmidt*

*German Research Center for Artificial Intelligence
PO Box 2080
D-6750 Kaiserslautern
e-mail: kuehn,schmidt@dfki.uni-kl.de

+GMD
Expert System Research Group
PO Box 1240
D-5205 St. Augustin 1
e-mail: linster@gmdzi.gmd.de

Abstract

For a simplified version of the clamping tool selection problem in mechanical engineering, the knowledge acquisition tool COKAM is applied to obtain an informal knowledge base and explanation structures from technical documents and prior case solutions. The output of COKAM is used to construct a three layered KADS conceptual model, which is then transformed into an operational model in the language OMOS. The OMOS formalization allows to verify the informal KADS conceptual model and to check the completeness of the domain knowledge. The results of this analysis are utilised in the next knowledge elicitation session with COKAM.

1. Motivation

This paper describes how three tools and approaches, each representing a different contribution to knowledge acquisition, have been used in one application to obtain qualitatively new knowledge acquisition potential. COKAM [Sc90a] is a knowledge elicitation tool. KADS [Br89] is a systematic knowledge engineering approach. OMOS [Li90] is a representation language for operational models of problem-solving. The interaction of the tools and approaches (see Figure 1.1) looks as follows:

- The output of the knowledge elicitation tool COKAM, which produces informal knowledge units and explanation structures from technical documents and from prior case solutions, is transformed into a KADS conceptual model.

- The KADS conceptual model serves as an explanation framework in the ensuing analysis of COKAM-generated explanations. Thus the conceptual model is tested against the next cases. This validates the model in the continuing elicitation.
- The structure-preserving operationalisation of the KADS conceptual model in the representation language OMOS enforces an unambiguous definition of the knowledge of the conceptual model. A reduction of ambiguity, vagueness and a better understanding of concepts can be achieved that way ([Ak90], p. 9).
- The cases, that were the basis for the COKAM explanation structures can be used to validate the OMOS model.
- The OMOS analysis features point out loopholes in the OMOS knowledge base. Because of the structure-preserving transformations they correspond to loopholes in the KADS conceptual model, which again are due to loopholes in the COKAM case and explanation base.

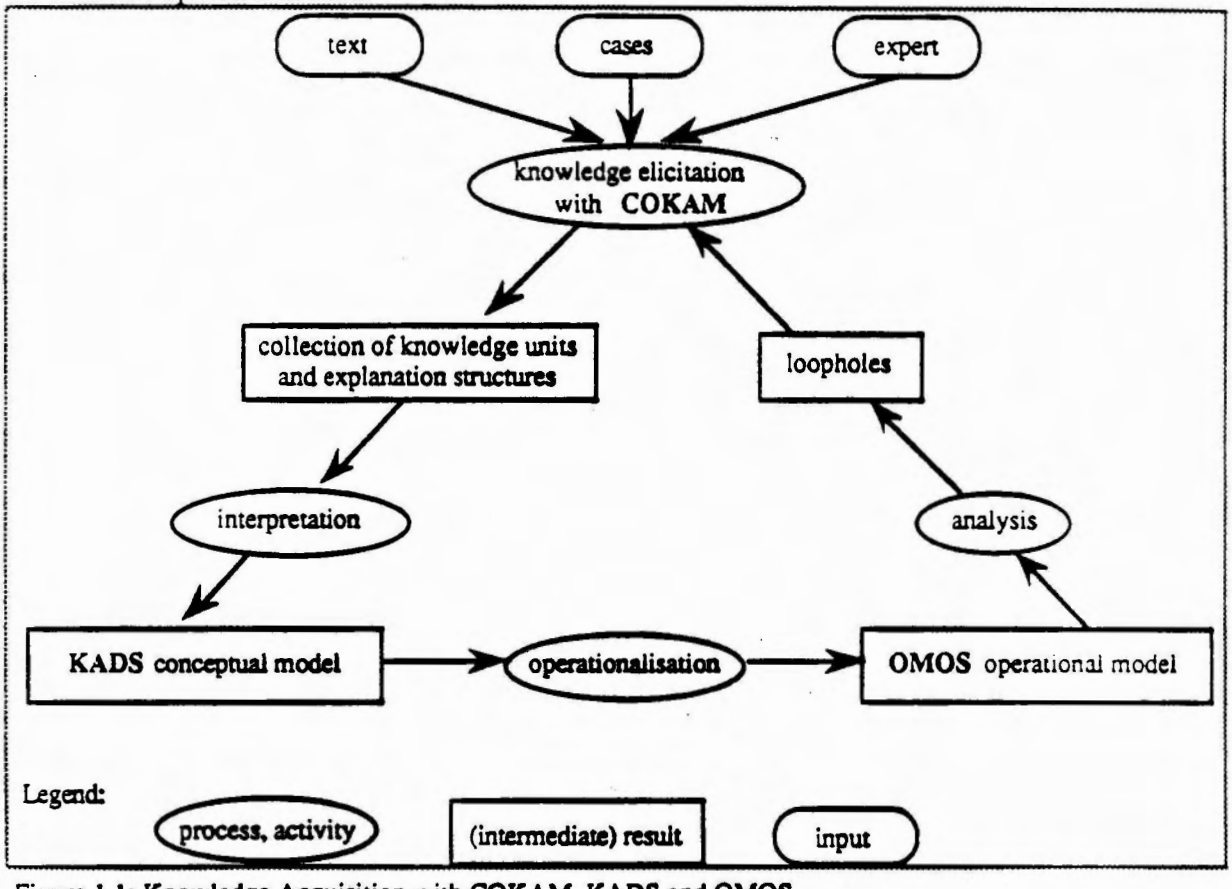


Figure 1.1: Knowledge Acquisition with COKAM, KADS and OMOS

Section 2 of this paper describes our application: the selection of an appropriate clamping tool to fix a workpiece for lathe turning. The third section describes how the tool COKAM is applied to elicit knowledge and explanation structures from texts and prior case solutions. In Section 4 we will show how the output of COKAM is used to develop an adequate KADS conceptual model. Section 5 demonstrates how the KADS conceptual model is operationalised with the representation language OMOS, and Section 6 shows how the analysis tools of OMOS provide useful hints for the use of COKAM to complete the elicited knowledge. Section 7 summarises and discusses the COKAM-KADS-OMOS approach.

2. Our Application Domain: Clamping Tool Selection for Lathe Turning

COKAM was developed in the ARC-TEC project (Acquisition, Representation and Compilation of TEChnical Knowledge; [Ri89]). The goal of this project is the development of a domain specific shell for the construction of expert systems that solve various tasks in mechanical engineering. One prototypical task is the generation of a production plan for a rotational part. The production plan specifies:

- the clamping tool with which the workpiece is to be fixed in the lathe
- the cutting tools with which the material is to be removed from the mold
- the sequence of cuts
- the cutting parameters such as feed and revolutions per minute.

Figure 2.1 shows a typical rational part (a drive shaft) overlaid with the mold from which it is to be manufactured. The numbers indicate the sequence of cuts by which the material is removed. They also refer to concrete cutting tools. Since the whole task of production planning is rather complex, we will only consider some aspects of the clamping tool selection problem to demonstrate our approach.

The clamping tool serves to centre the workpiece in the lathe and to transmit the rotation. In Figure 2.1, it is depicted by the two black parts labeled left fixture and right fixture. In this particular case the left fixture, which transmits the rotation is a lathe dog and the right fixture is a lathe centre. Other types of clamping tools are clamping jaws and collet chucks.

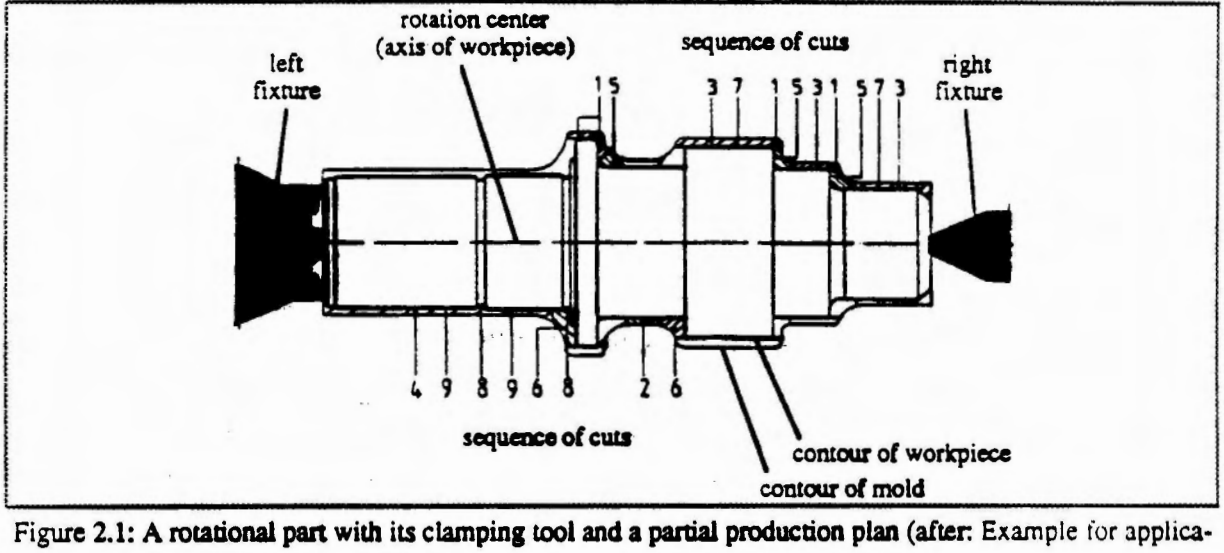


Figure 2.1: A rotational part with its clamping tool and a partial production plan (after: Example for application, SPK-Feldmühle Werkzeuge, [We, undated])

Lathe dogs hold the workpiece from the side and allow free access to all surfaces except the left and the right vertical plane. Clamping jaws and collet chucks, on the other hand, do not use the vertical planes to fix the workpiece but use instead parts of the outside plane (represented by horizontal lines in Figure 2.1). Depending on the turning requirements, i.e., which surfaces of the workpiece are to be manufactured, either a lathe dog or collet chucks are more advantageous. Besides the accessibility of different sections of the workpiece surface there are several other criteria that are relevant for the selection of a clamping tool. In our application we will only consider the set-up time, i. e., the time that is needed to mount the

clamping tool on the lathe and the clamping time, the time it takes to close the clamping tool when a new mold is inserted into the lathe.

The selection of an appropriate clamping tool depends not only on the geometry and the technological requirements of the workpiece to be manufactured but also on the available production environment (machines and already mounted tools) and the customer´s requirements (lot size and delivery deadlines).

In the next two sections of the paper we will develop a KADS conceptual model for this simplified version of the clamping tool selection problem. We construct a KADS conceptual model with the help of the knowledge acquisition tool COKAM. We will only describe the subset of the functionality of COKAM which is relevant for the construction of the conceptual model. A more comprehensive description is given in [Sc90a and Sc90b].

3. Knowledge Acquisition with COKAM

3.1. Basic Principles

COKAM (Case-Oriented Knowledge Acquisition Method from Text) is based on the idea that different traces of expertise which serve as information sources should be used together for knowledge acquisition. In the domain of mechanical engineering there are three different traces of expertise: namely texts, records of previously solved cases, and expert's know-how. These three traces of expertise complement each other since to some extent they provide different information. Even when they contain the same information the knowledge may be obtained more easily from one information source than the other. Therefore, integrated knowledge acquisition [Sc90b] in which different traces of expertise are used, offers significant advantages. Since the traces supply different and possibly overlapping knowledge, the completeness and consistency of a constructed knowledge base can more easily be established. Furthermore the relevant knowledge can be elicited more efficiently when several information sources are used.

COKAM combines the three traces of expertise as follows: Texts are the primary information source since they contain general and well structured knowledge in explicit form. Segments that are extracted from the text by an expert constitute the building blocks or knowledge units of the initial informal knowledge base. Although texts are used first, records of previously solved cases are just as important for knowledge acquisition with COKAM. Cases are used to obtain information of how the mostly declarative knowledge that was extracted from the text is applied by the expert in practice. In COKAM this is accomplished by having the expert explain solutions of different cases using the knowledge units in the informal knowledge base. Cases are used together with their solutions, since it is less time consuming to explain than to generate a solution of a particular case. The explanations of individual cases can be analyzed to check the relevance and sufficiency of the informal knowledge base. It is even more interesting, however, that the obtained explanation structures can be used to determine the task and inference structure of a KADS conceptual model.

3.2. Extracting an Initial Informal Knowledge Base from Text

Figure 3.1 shows how COKAM supports the construction of an initial informal knowledge base from a technical text. The text (which can be scanned in) is presented on the left of the screen. The text browser, which is located directly above the text window, helps the expert

navigate through the text. The right side of the screen shows the informal knowledge base. Small windows display the knowledge units, which are arranged in the form of a card stack. The user can move up and down the stack to select a particular knowledge unit. The selected unit is always located next to the dialogue window in the centre of the screen. COKAM supports copying text segments to a knowledge unit, editing a knowledge unit, and assigning it to various categories. It records the history of each knowledge unit so that the user can at any time go back to the piece of text from which it originated.

The text that was used in our application contained different types of information. It described the different types of clamping tools, the function of a clamping tool, the criteria for selecting an appropriate clamping tool, and how these criteria are applied in practice. The expert was told to select all the text passages that contain relevant information for the solution of the target task, since only then a detailed explanation at different levels of abstraction can be given. The knowledge units that were thus constructed usually consisted of single text sentences, of partial sentences or combinations of two sentences. These text fragments were modified so that they could be understood without their context, i.e., anaphora and other references were resolved. Table 3.1 shows a selection of knowledge units that were obtained from our text. The expert then used these knowledge units to construct explanations for individual cases.

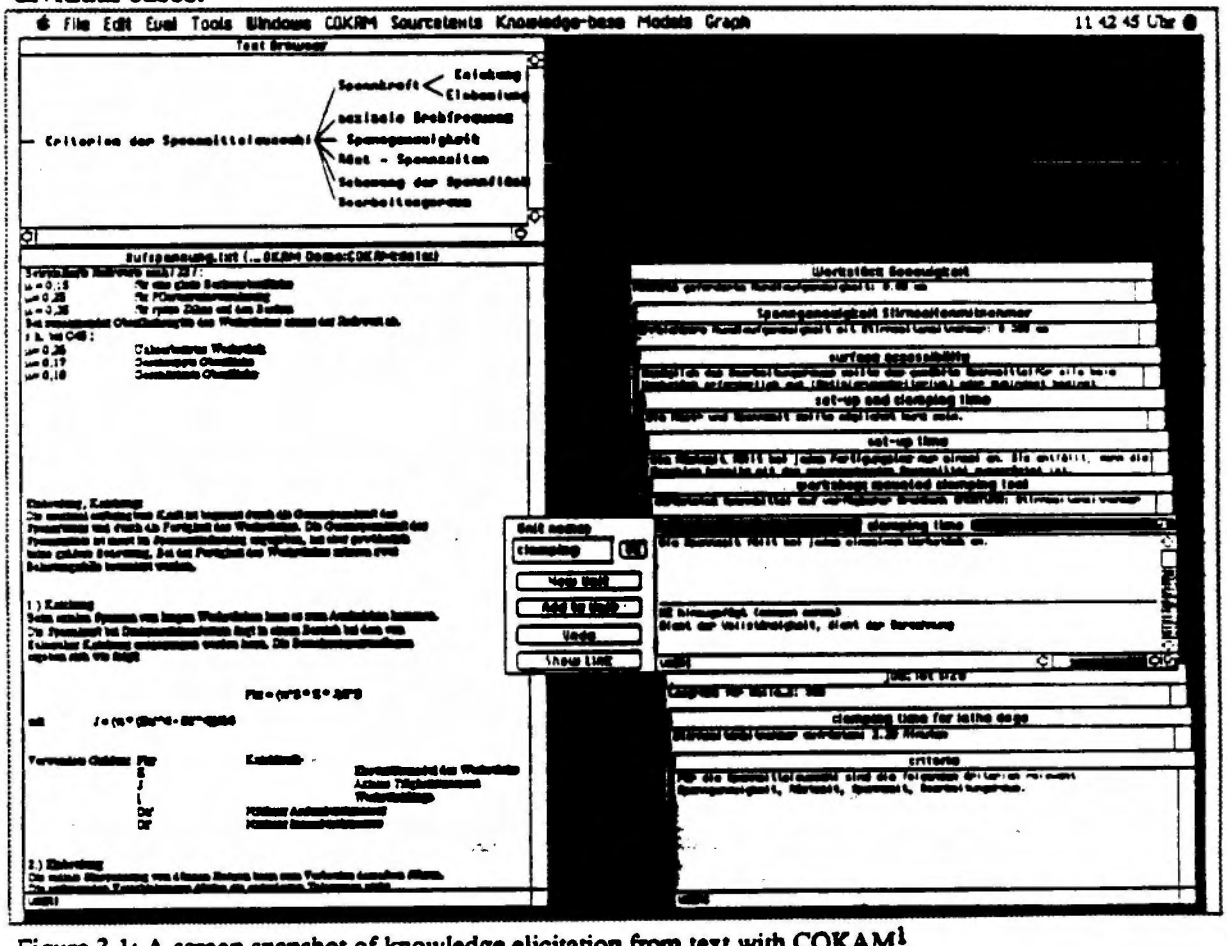


Figure 3.1: A screen snapshot of knowledge elicitation from text with COKAM[1].

[1] A German text was used for knowledge elicitation, as our experts are German mechanical engineers.

Table 3.1: A sample of knowledge units of the informal knowledge base.

1. "The purpose of the clamping tool is to centre the workpiece in the lathe and to transmit the rotation."
2. "For the selection of a clamping tool the criteria accessibility of workpiece surfaces, clamping time and set-up time must be considered."
3. "A lathe dog is ideal for transverse turning but axial turning is also possible."
4. "The set-up time for a lathe dog is 3 Minutes."
5. "The set-up time of a selected clamping tool should be as short as possible."

3.3. Obtaining Explanations for Individual Cases

The screen set-up for constructing explanations is similar to Figure 3.1, except that instead of the text window an explanation window is displayed, which initially contains a case description. We build an explanation tree by selecting knowledge units from the informal knowledge base and attaching them to the case description or to other units in the explanation window. When constructing the explanation, one can add new units to fill gaps in the informal knowledge base. Existing knowledge units can be reformulated, so that they can be better understood. Constructing an explanation for a case thus also helps to get a more precise formulation of the units.

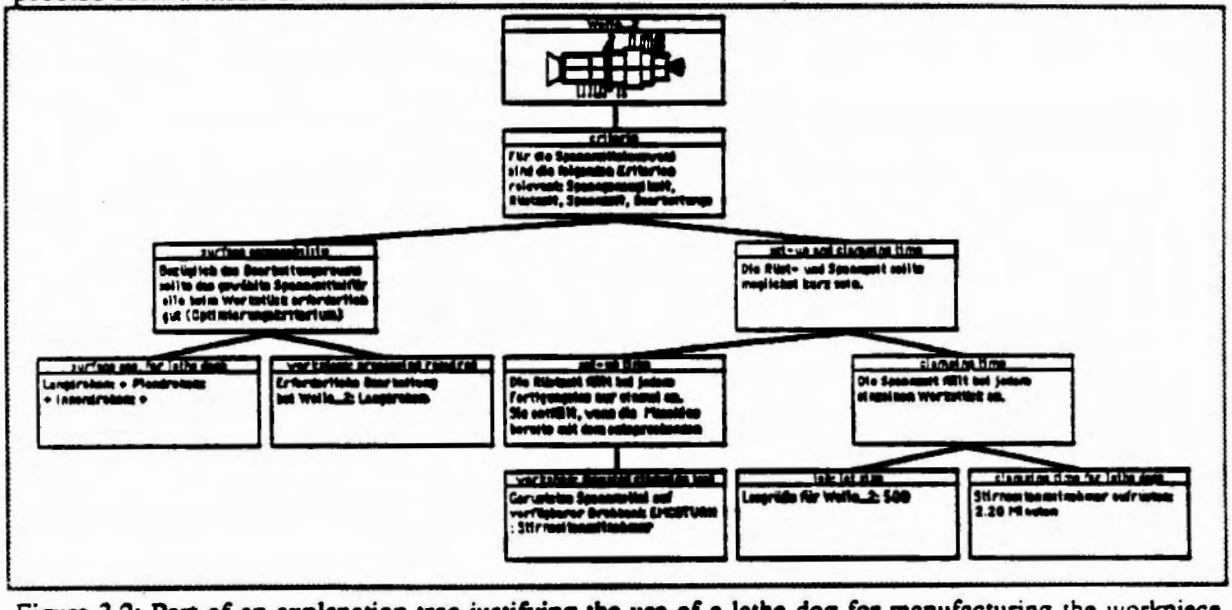


Figure 3.2: Part of an explanation tree justifying the use of a lathe dog for manufacturing the workpiece "Welle_2."[2]

Figure 3.2 shows a part of a typical explanation tree. The knowledge unit directly below the graphical representation of the case names the criteria for clamping tool selection (see knowledge unit 2 of Table 3.1). The subsequent units show how these criteria are satisfied for the particular problem. The leaves of the explanation tree always refer to concrete

[2]For the reader's convenience, the knowledge units containing German text are given English labels which characterize their contents.

elements of the case description, i.e., the workpiece data, the job data, the workshop data or the given solution. The links between the knowledge units should be interpreted as "explain" links, i.e., the units below a particular unit together explain that unit.

4. Constructing the KADS Conceptual Model

4.1. The Subset of KADS that We Consider

KADS (Knowledge Acquisition and Design Structuring; [Br87, Sc88 and Wi89]) is a methodology for the structured and systematic development of knowledge-based systems. KADS divides the development process of a knowledge-based system into a suite of models (in particular the conceptual model, the design model, the logical model; [Sc90c]) that represent the transformation of knowledge from the initial, non-formal phases into an operational system. Within the models, knowledge is represented on several layers to stress the different types of knowledge of a knowledge-based system.

Our major concern is the development of the conceptual model. According to our interpretation the conceptual model is a model of the expert's problem-solving process. It should also be the basis for the design of the system. For the construction of the conceptual model we will rely on the informal knowledge base and the explanation structures collected with COKAM. According to the KADS terminology we will distinguish a domain layer, an inference layer and a task layer. The domain layer describes the static domain knowledge, which can be used for various tasks. Entities in the domain layer are concepts and relations. The inference layer specifies what inferences can be made in terms of knowledge sources and meta-classes. Knowledge sources are functional descriptions of inference making and meta-classes describe the role that the domain layer concepts can play. The inference layer is neither domain-specific nor task-specific since the knowledge sources and meta-classes may occur in different domains and different tasks. The inference structure for a specific task, however, specifies only that part of that inference layer that is relevant for the solution of this task. The task layer describes how the knowledge sources of the inference layer are applied to solve a particular task. The task structure describes one particular way to solve a task. The strategy layer controls the selection of a task solution method. We do not consider this layer since we did not observe different strategies in our application task.

4.2. The Domain Layer

The concepts and relations that describe the static knowledge of the application domain can be identified from the elements of the informal knowledge base. Since the knowledge units contain decontextualised pieces of information, which were considered to be relevant by the expert, it is easier to discover concepts and the relations in the knowledge units than in complete texts or in interview protocols.

The informal knowledge base that was constructed with COKAM contains many knowledge units that describe relations between domain concepts. For example the knowledge unit 3 of Table 3.1, which states that a *lathe dog* is an *optimal clamping tool* for *transverse turning* and a possible clamping tool for *axial turning*, or more technically, the relation between *lathe dog* and *transverse turning* is *optimal clamping tool* and the relation between *lathe dog* and *axial turning* is *possible clamping tool*. Another example is knowledge unit 4 of Table 3.1, which states that the *set-up time* for a *lathe dog* is 3 minutes.

The relevant domain concepts for clamping tool selection that were identified from the units of the informal knowledge base are:

- *clamping tool* and the various kinds of clamping tools (e.g., *clamping jaw*, *lathe dog* and *collet chuck*),
- *surface accessibility* which depends on the *turning requirements*: *transverse turning*, *axial turning*, *inside turning*,
- *clamping time*, *set-up time*,
- *workpiece*.

The relevant domain relations are:

- *required processing* and *possible processing*,
- *has clamping time* and *has set-up time*,
- *mounted clamping tool*,
- *ideal clamping tool* and *feasible clamping tool*.

4.3. The Inference and Task Layers

We will not separate the discussion of the inference and task layer. They both describe the problem-solving method and depend strongly on each other. The identification of the expert's problem-solving method is a major difficulty in knowledge acquisition. One approach to overcome this difficulty is the collection of all known problem-solving methods in libraries that the knowledge engineer can consult. The KADS interpretation model library [Br87], Generic Tasks [Ch86] and the collection of role-limiting methods [Mc88] pursue this goal. Even when comprehensive libraries exist, the selection of an appropriate model still constitutes a problem. Furthermore, we believe that such libraries can never be both exhaustive and provide sufficiently detailed models. This means, that only a global model can be selected from the library, which must be refined to fit the specific task. The knowledge acquisition method COKAM supports the knowledge engineer in the selection and refinement of an appropriate problem-solving model.

The problem-solving model that we will suggest for the task of clamping tool selection was not part of the KADS library of interpretation models. We had to build a new one. Psychological research on human decision making [Hu82 and Ge90] provides the foundation for our model.

When people are confronted with a task of selecting one alternative from a large number of given alternatives based on several criteria, they first reduce the set of alternatives by eliminating some obviously bad alternatives to obtain a smaller, more manageable set of alternatives, which are then examined in more detail. In order to obtain a manageable set of alternatives the selection criteria are applied more or less rigourously. In a second step, the subjective costs of the remaining alternatives are computed on the basis of optimization criteria and the best one is selected. The described problem-solving method, which might be termed *selection by elimination and optimization*, has been observed when people select an apartment to rent or a car to buy.

The problem of selecting a clamping tool is similar to the above problems in that there are a large number of solution alternatives (not illustrated in the example of this paper) and several criteria that make an elimination process necessary. The data collected with COKAM

show that the above problem-solving method adequately describes how the expert solves the clamping tool selection problem.

4.3.1. The Inference Structure

A closer inspection of the units of the informal knowledge base shows that they not only describe relations of domain concepts but also refer to inference layer elements. The inference layer units can be identified by searching the knowledge units for keywords that are typically used to name meta-classes and hardly ever refer to domain concepts. Such keywords are: criterion, solution, hypothesis, alternative, feature, etc. For example, knowledge unit 2 of Table 3.1 contains the word *criteria*, and it indeed relates the domain concepts surface accessibility, clamping time and set-up time to the meta-class *criterion*.

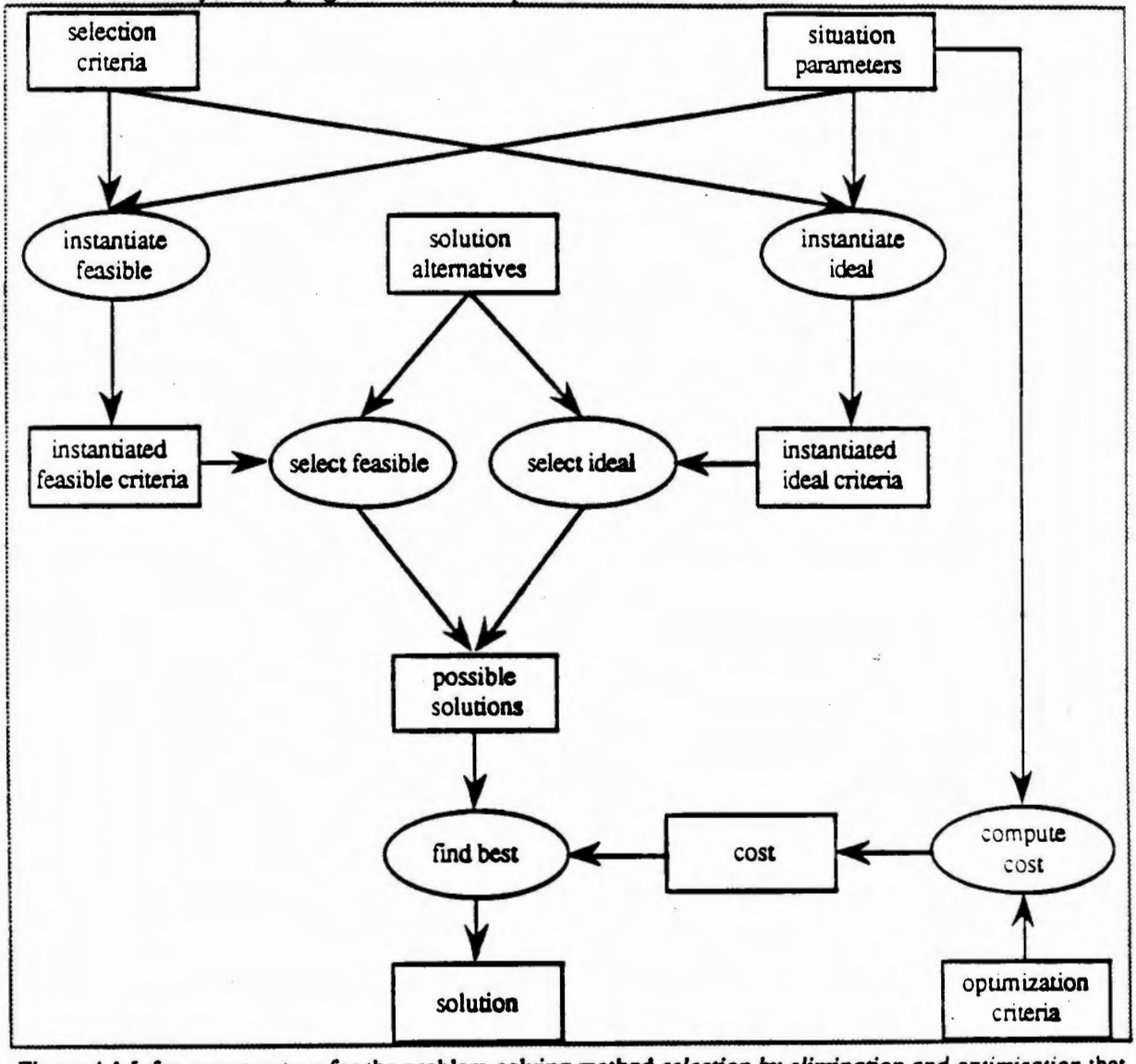


Figure 4.1 Inference structure for the problem-solving method *selection by elimination and optimization* that we use for clamping tool selection.

Besides the individual knowledge units that give some hints for possible meta-classes and knowledge sources, the construction of the inference layer is facilitated by the explanation collected with COKAM. Whereas the knowledge units at the leaves of the explanation

tree refer to the domain layer, the inner knowledge units pertain to the inference layer. There are three such knowledge units in the explanation tree shown in Figure 3.2, namely the units directly below the problem description.

When the explanations that were generated with COKAM for different cases are compared to each other, it can be seen that the criteria are usually applied in the same order, i.e., the criterion *surface accessibility* is considered before the criteria *set-up* and *clamping time* (see the explanation structure in Figure 3.2). Therefore it is likely that *surface accessibility* is a *selection criterion* whereas *set-up time* and *clamping time* are *optimization criteria*. An inspection of the corresponding knowledge units confirms this interpretation. The selection criterion *surface accessibility* can be instantiated in two ways so that either only the ideal clamping tools (those clamping tools that are well suited for the required types of processing) or all feasible clamping tools (those clamping tools that are less suitable, but that still do the job) are selected.

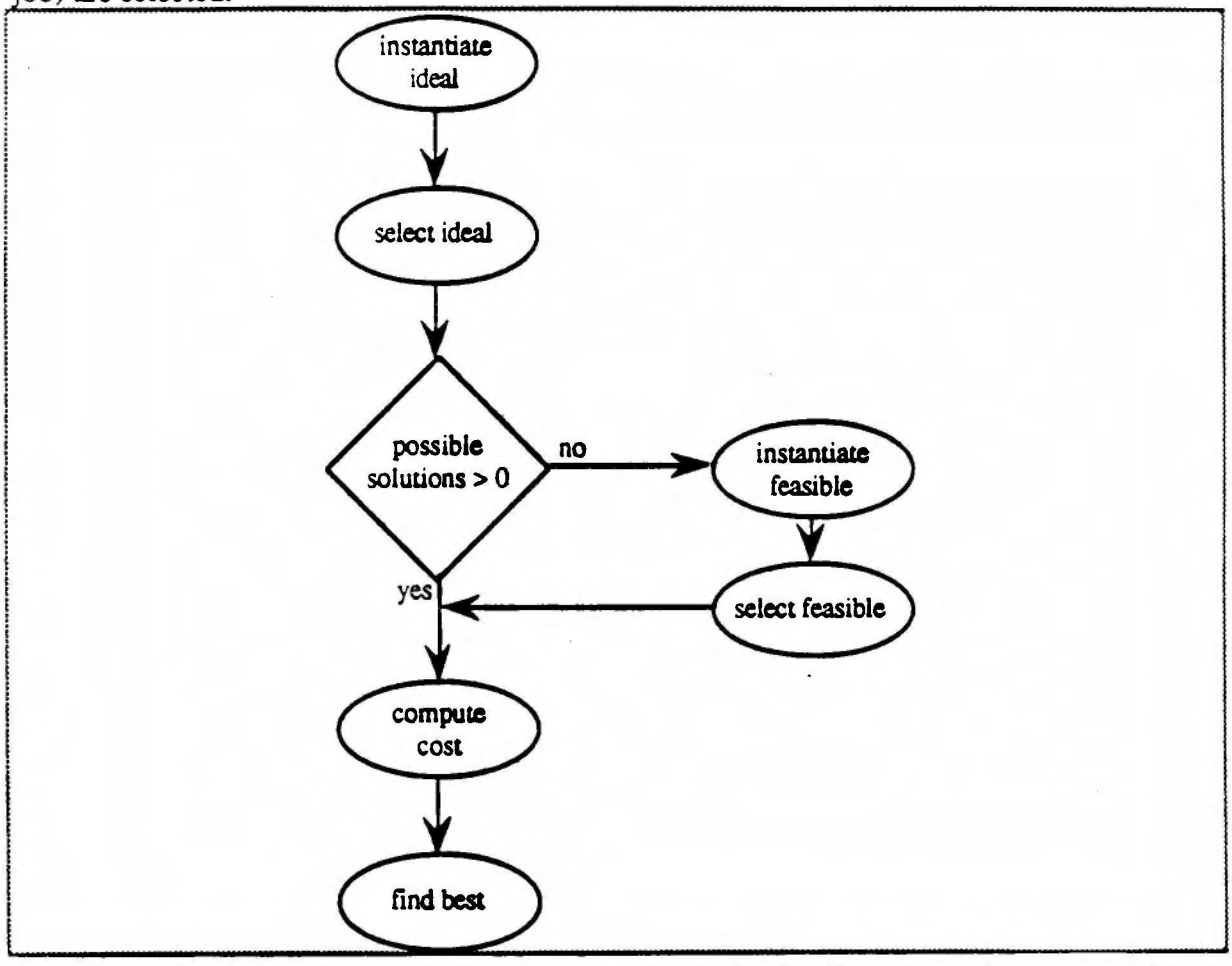


Figure 4.2: Task structure for the problem-solving method *selection by elimination and optimization* that we use for clamping tool selection.

Figure 4.1 shows the inference structure for clamping tool selection. There are two ways to select the possible solutions from the set of solution alternatives (e.g., the available clamping tools specified in workshop data). The selection criteria (surface accessibility) can be instantiated with the situation parameters (the problem description except the solution alternatives) so that either all feasible or only the ideal possible solutions are selected. From the set of possible solutions the best solution is the one with the lowest cost. The cost for

each possible solution is computed based on the optimization criteria (set-up and clamping time) and the relevant situation parameters (lot size and mounted clamping tool).

4.3.2. The Task Structure

The task structure describes the sequence in which the knowledge sources of the inference structure are used to solve the task. There are no units in the informal knowledge base that contain such information. This is not surprising since texts usually do not describe in detail how to solve particular tasks. To some extent the explanations that were generated with COKAM provide information about the sequence of knowledge sources as already pointed out in the previous section.

Figure 4.2 shows the task structure for clamping tool selection. The ideal instantiation of the selection criteria is used first, since it allows a more extensive reduction of the number of solution alternatives. If the ideal instantiation does not produce any solutions, then we resort to feasible, sub-optimal instantiations of the selection criteria. In our application, the selection criteria are descriptions of the surface accessibility required to turn a workpiece: inside, transverse or axial turning. For the solutions of the selection process we compute costs. Cost computation uses optimization criteria, which in our application correspond to set-up time and clamping time.

5. Operationalizing the Conceptual Model with OMOS

This section of the paper describes how the KADS conceptual model of the clamping tool selection task is represented in the operational language OMOS and which transformation processes are necessary to do so. We will limit ourselves to a brief description of the language OMOS and of the implemented clamping-tool-selection system, as this article focusses on the feedback that a structure-preserving implementation of an explicit model of expertise provides for the continuing knowledge elicitation and analysis (see Figure 1.1).

5.1. The Language OMOS

OMOS (Operational Models of Problem-Solving) is a representation language for operational models for knowledge-based systems [Li90]. OMOS models consist of a generic part that describes the problem-solving method, of an application-specific part that describes the domain knowledge that is used by the problem-solving method and of a mapping that describes which domain element is used by which element of the problem-solving method.. OMOS tries to combine the advantages of KADS (i.e., flexibility of modelling for a wide range of tasks, reusability of the generic aspects of the models) with the advantages of operational approaches to the modelling of problem-solving as we find them in PROTEGE/p-OPAL (i.e., use of operational data-structures to represent detailed domain-knowledge, automatic analysis capabilities and strong guidance in the acquisition and representation of the application-specific knowledge; [Mu89, We90 and Vo90] describe related work.

OMOS represents knowledge in a two-layered KADS-like framework, consisting of a domain layer and of a layer for the problem-solving method.

5.1.1. The Language Constructs for the Domain Layer

On the domain layer OMOS provides language facilities for the definition of concept hierarchies and for the definition of relations between concepts.

Frames, attribute-descriptions for the frames and instances of the frames are the building blocks for the concept hierarchies. In OMOS, frames are used to define domain types (e.g., the type *clamping-tool).* Frames can be included in multiple-inheritance hierarchies. Attributes, defined for the frames, provide uniform descriptions for the instances of the frames, such as the attribute ***set-up-time*** for the frame *clamping-tool.* Concepts of the domain layer are represented as instances of frames with values for the attributes.

Relations express information that relates to several concepts. The relation *optimal-clamping-tool* relates clamping tools (i.e., instances of the frame *clamping-tool*) to turning requirements (i.e., instances of the frame *turning-requirement*). Relations are lists of tuples or predicates, whose extension corresponds to a finite list of tuples. The arguments of the relations correspond to instances of frames. The relation *optimal-clamping-tool* has instances of the frame *clamping-tool* as a first argument whereas the second argument refers to instances of the frame *turning-requirement.*

5.1.2. The Language Constructs for the Layer of the Problem-Solving Method

On the layer of the problem-solving method OMOS provides inference actions, roles, inference structures and control-structures to represent a problem-solving method. OMOS represents a problem-solving method in KADS-oriented ways: the knowledge sources of KADS have inference actions as their counterparts; the meta-classes are paralleled by roles in OMOS. We use different terms for these basically similar concepts to state clearly that we do not have the prention to implement KADS, and because KADS uses the inference and task layer elements to conceptualise problem-solving behaviour, whereas OMOS uses its terms as an implementation language.

Roles describe parts that domain layer concepts play in a problem-solving process. Initially the role *solution-alternative* describes the concept *collet-chuck.* In case it is selected because the workpiece could be turned using a *collect-chuck* then it changes its role and becomes a *possible-solution.* If it is the best solution as far as some cost-factor is concerned, then it plays the role *solution.* Thus roles in OMOS are assigned dynamically.

Inference actions are functions that work on domain layer concepts. They can modify the role-assignment of a concept or they can change value-assignments of attributes of a concept. An OMOS inference action has one input role, one output role and several control roles. An inference action moves concepts from the input role to the output role. It can assign values to concepts that are currently denoted by its output role. Conditionals are expressed through the elements of the control roles. Inference actions use the knowledge that is expressed in the domain-layer relations. The inference functionality of an inference action is defined declaratively by stating the type of value-assignment it implements (none, initial or modification), its type of role-assignment (none or transfer), the domain-layer relation it uses and the input, output and control meta-classes.

The inference structure is defined implicitly through the roles and inference actions. It is not an explicit construct of OMOS.

The control structure specifies the sequence in which the inference actions are called. Conditionals may only refer to inference actions and roles. The control structure corresponds to the fixed task structures of KADS. OMOS does not provide an equivalent for the KADS strategy layer.

5.1.3. The Interpretation of the Language OMOS

OMOS is implemented in BABYLON [G87 and Pr87]. The domain layer elements of OMOS are BABYLON language constructs: frames and instances for the types and the concepts; PROLOG for the tuples of the relations. The constructs of the problem-solving method are compiled into BABYLON constructs. The roles are dynamic PROLOG predicates, that hold for pairs of domain-concepts and roles. The inference actions are transformed into forward-rule schemes, which implement the value-assignment and role-assignment description and that use the underlying domain relation. When an inference action is called, the rule-scheme executes for all the tuples of the domain relation. Thus all the constructs of OMOS are transformed into BABYLON constructs, and we only need a rudimentary interpreter.

5.2. Application Limitations for OMOS

The current implementation of OMOS is limited to finite domains. The problem-solving methods must obey several characteristics. The problem-solving method must consist of a limited number of inference actions, that are called-upon in a pre-defined iteration. Problem-solving roles may only refer to domain-concepts, and not to attributes of concepts or to tuples of relations. The problem-solving process does not create or delete new domain instances. Except for the last constraint, it is not clear yet how these limitations restrain the applicability of OMOS. It has been used successfully for the implementation of more complex systems such as K-ONCOCIN, a re-modelling of large parts of ONCOCIN with KADS terminology [Li91]. The last constraint is severe, as the current version of OMOS cannot be used for the modeling of synthesis problems that require the dynamic creation of solution elements at run-time.

5.3. Formalizing the Conceptual Model of Clamping Tool Selection

5.3.1. The Domain Layer

The domain language features of OMOS allow for a straight-forward implementation of the domain layer knowledge of the KADS conceptual model described in Section 4.2.

5.3.1.1. The Concepts

Figure 5.1 shows the hierarchical definition of types and instances for the domain layer concepts. Not all the concepts described in Section 4.2 appear in Figure 5.1, as not all of them have been defined explicitly in OMOS (e.g., *lot-size* is a concept of the conceptual model, but an attribute with values in the implementation). Several of the relations described in the conceptual model were implemented as attributes of concepts too, for example *has-clamping-time*, *mounted-clamping-tool*, *required-processing*. Thus they do not appear in Figure 5.2.

In the implementation we had to introduce the type *workpiece-cost-element* (see Figure 5.1) to provide concepts that can play the roles defined by the meta-class *optimization-criterion*. This is because OMOS cannot apply roles to attributes, only to concepts. The type *turning-requirement* is a more restrictive, and within the context of our system much more precise term for *surface-accessibility*. Its instances describe precisely which kind of turning operation is needed to model the surface of a workpiece, i.e., how we have to access the surface with our tool.

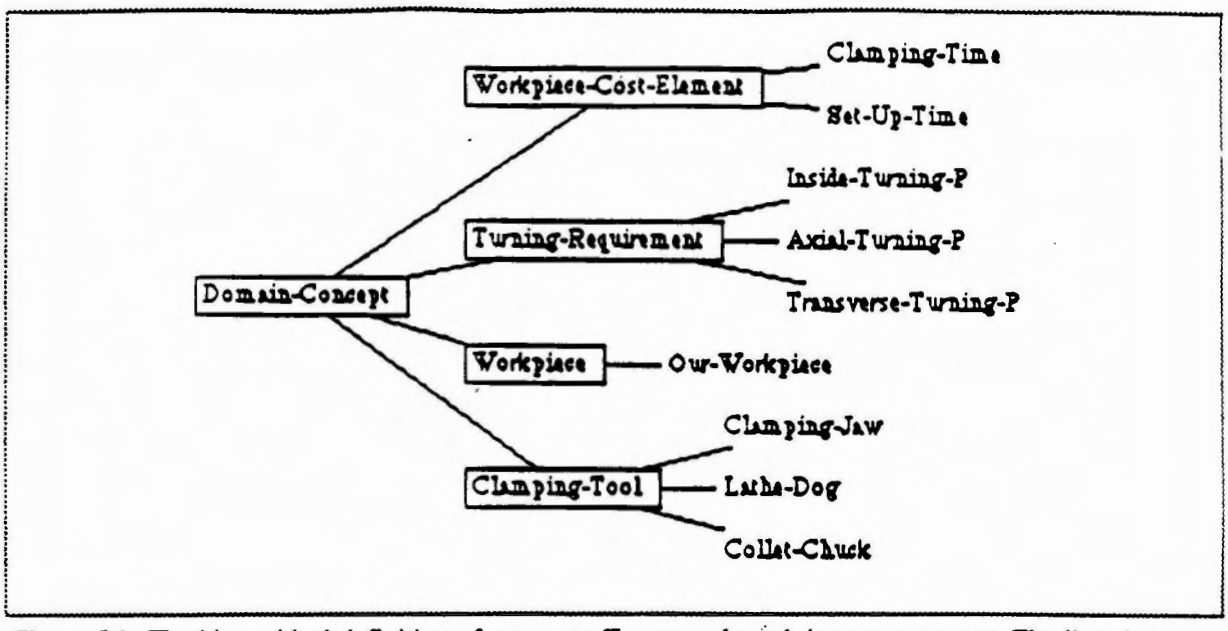


Figure 5.1: The hierarchical definition of concepts. Types are boxed, instances are not. The lines between types denote inheritance. The lines between types and instances describe instantiation.

5.3.1.2. The Relations

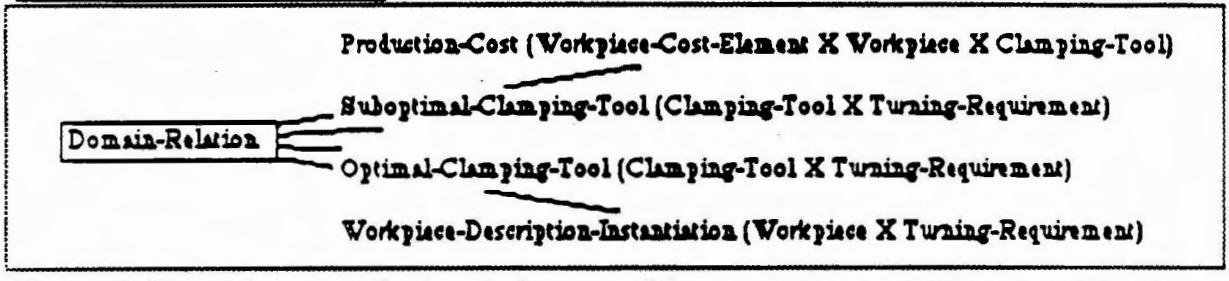


Figure 5.2: The relations are defined on the instances of the domain layer types. The tuples of the relation *production-cost* have instances of the type *workpiece-cost-element* in the first argument position, instances of *workpiece* in second position and the last argument is of type *clamping-tool*.

Table 5.1: The definition of the domain layer relation *optimal-clamping-tool* and some of the tuples of the relation.

```
(DEFINE-DOMAIN-RELATION Optimal-Clamping-Tool
 WITH ARITY = 2
      TYPE-SEQUENCE = (((INSTANCE Clamping-Tool)
                        (EXTENSION-* Turning-Requirement)))
      ARGUMENT-SEQUENCE =
                       ((Clamping-Tool Important-Turning-Requirements))
      TUPLES = ((
                ((Optimal-Clamping-Tool
                     Collet-Chuck
                          ((Transverse-Turning-P Required-Value = Yes)
                           (Inside-Turning-P Required-Value = No))))

                ((Optimal-Clamping-Tool
                     Lathe-Dog
                     ((Axial-Turning-p Required-Value = Yes)
                      (Inside-Turning-P Required-Value = No))))
```

Table 5.2: The definition of the domain layer relation *suboptimal-clamping-tool* and some of the tuples of the relation.

```
(DEFINE-DOMAIN-RELATION Suboptimal-Clamping-Tool
 WITH ARITY = 2
      TYPE-SEQUENCE = (((INSTANCE Clamping-Tool)
                        (EXTENSION-* Turning-Requirement)))
      ARGUMENT-SEQUENCE =
                      ((Clamping-Tool Important-Turning-Requirements))
      TUPLES = ((
                      ((Suboptimal-Clamping-Tool
                           Collet-Chuck
                           ((Inside-Turning-P Required-Value = No))))

                      ((Suboptimal-Clamping-Tool
                           Lathe-Dog
                           ((Inside-Turning-P Required-Value = No))))
...
```

The relation *optimal-clamping-tool* (see Table 5.1) describes which clamping tool is a first choice for certain turning requirements, for example a *collet chuck* is the optimal choice if the workpiece requires transverse turning and inside turning is not needed. The relation *suboptimal-clamping-tool* (see Table 5.2) describes a relaxation of the relation *optimal-clamping-tool*. For example the second tuple *(Suboptimal-Clamping-Tool Lathe-Dog ((Inside-Turning-P Required-Value = No)))* is a formal representation of the third knowledge unit of Table 3.1: *a lathe dog is ideal for transverse turning but axial turning is also possible.* In the formal representation of the relaxed choice criteria, we only mention that a lathe dog can definitely *not* be used for inside turning.

5.3.2. Formalization of the Problem-Solving Method *Selection by Elimination and Optimization*

The problem-solving method of the operational system is built from the inference and task layers of the KADS conceptual model. KADS knowledge sources are implemented with OMOS inference actions; meta-classes are represented as roles.

5.3.2.1. Modifications to the Inference Structure of the Conceptual Model

The formalization of the underlying domain layer knowledge has shown us that the knowledge sources *instantiate-feasible, instantiate-ideal, select-ideal* and *select-feasible* are redundant in the problem-solving method *selection by elimination and optimization.* It is sufficient to instantiate the selection criteria once and then use them in two different ways. Furthermore we noticed that the meta-classes *cost* and *possible-solutions* cannot be separated, as they both relate to different aspects of the same domain concepts[3]. Thus in the OMOS inference structure the inference action *compute-cost* operates on the role *possible-solution.* In OMOS roles describe the part that concepts play and they do not refer to a set of concepts currently playing that part. Thus the set-like descriptors of the KADS conceptual model, such as *possible-solutions* become role-descriptors such as *possible-solution,* which will be used in the OMOS inference structure (see Figure 5.3), which is a simplification of the inference structure of the KADS conceptual model (see Figure 4.1).

---

[3] This is due to a limitation of the language OMOS because only concepts can play roles and not attributes of concepts.

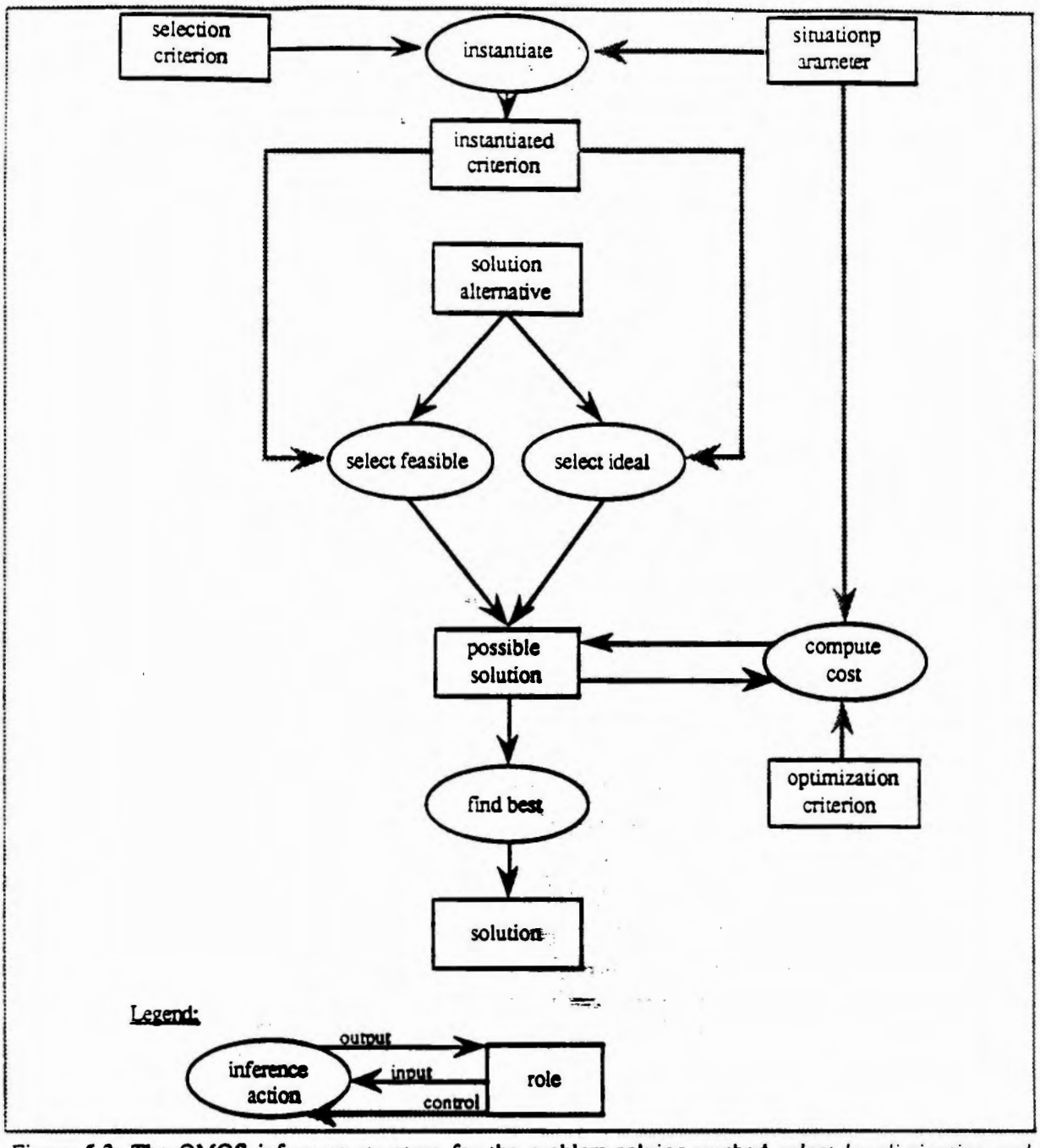


Figure 5.3: The OMOS inference structure for the problem-solving method *select by elimination and optimization*.

5.3.2.2. The Inference Actions and the Roles

Role-assignments to concepts are dynamic in OMOS. Some concepts have initial roles; for example all the instances of the type *clamping-tool* have the initial role assignment *solution-alternative* (see Table 5.3). The current role-assignment is part of the current state of the problem-solver.

Table 5.3: The definition of the role *solution-alternative*, with its initial role assignment referring to all the instances of the domain type *clamping-tool*.

```
(DEFINE-ROLE Solution-Alternative
  WITH INITIAL-ROLE-ASSIGNMENT = ((Clamping-Tool _Tool)))
```

Table 5.4: The definition of the inference action *select-ideal* with its corresponding roles, the value- and role-assignment descriptions, the domain layer relation and the mapping of the roles onto the arguments of the domain layer relation.

```
(DEFINE-INFERENCE-ACTION Select-Ideal
 WITH INPUT-ROLE = Solution-Alternative
      OUTPUT-ROLE = Possible-Solution
      CONTROL-ROLES = ((Instantiated-Criterion))
      VALUE-ASSIGNMENT = FALSE
      ROLE-ASSIGNMENT = TRANSFER
      DOMAIN-RELATION = Optimal-Clamping-Tool
      REL-ARG-TYPE = (((0 . INSTANCE)(1 . EXTENSION-*)))
      PROJ-ROLES-REL-ARGS = (((INPUT-ROLE . 0)
                              (OUTPUT-ROLE . 0)
                              ((CONTROL-ROLES . 0) . 1))))
```

In OMOS inference actions are defined through their roles, the domain relation they use, the value- and role-assignment descriptions and the mapping of the roles onto the argument positions of the underlying domain relation. A compiler translates this definition (see Table 5.4) into a forward rule-schema. Table 5.5 shows how the first tuple of the relation *optimal-clamping-tool* (see Table 5.1) is used as a forward chaining rule to implement the inference action *select-ideal* (see Table 5.4).

Table 5.5: A beautified version of the forward-chaining rule that the compiler generates from the definition of the inference action and from the first tuple of underlying domain relation *optimal-clamping-tool*.

```
IF      (CURRENT-ROLE Solution-Alternative Collet-Chuck)
        (CURRENT-ROLE Instantiated-Criterion Transverse-Turning-P)
        (CURRENT-ROLE Instantiated-Criterion Inside-Turning-P)
        (Transverse-Turning-P Required-Value = Yes)
        (Inside-Turning-P Required-Value = No)
THEN    (TRANSFER-ROLE Solution-Alternative Possible-Solution Collet-Chuck)
```

5.3.2.3. The Control Structure

The control structure for the problem-solving method *selection by elimination and optimization* (see Figure 5.4) is identical to the task structure of the KADS conceptual model (see Figure 4.2), except for the changes described for the OMOS inference structure. In OMOS control structures, conditionals may only refer to elements of the inference structure. In Table 5.6 the statement `(CALL-IA Instantiate)` is an activation of the inference action *instantiate*. `(ROLE-P Possible-Solution)` is a test that evaluates to true if at least one concept currently plays the role denoted by *possible-solution*.

Table 5.6: The control structure consists of a procedural definition of activations of inference actions (CALL-IA) and tests of role-assignments (ROLE-P).

```
(DEFINE-CONTROL-STRUCTURE Select-by-Elimination-and-Optimization
      CALL-SEQUENCE = (((CALL-IA Instantiate)
                        (CALL-IA Select-Ideal)
                        (WHEN (NOT (ROLE-P Possible-Solution))
                           (CALL-IA Select-Feasible))
                        (CALL-IA Compute-Cost)
                        (CALL-IA Find-Best))))
```

5.4. Insights through Formalization

As described previously, the formalization of the inferences has shown us that several knowledge sources of the inference structure of the KADS conceptual model (*instantiate-feasible*, *instantiate-ideal*, *select-ideal* and *select-feasible)* were redundant, even though our analysis of the explanation structures seemed to reveal that experts used different selection criteria in different selection processes. In retrospect though, this simplification is not a contradiction to the human problem-solving process described in Section 4.3.

6. Feedback from the Operational Model for the Knowledge Elicitation

6.1. The Use of the OMOS Analysis Capabilities for Focussed Knowledge Acquisition

OMOS provides a series of analysis tools to check the domain layer knowledge and the use of the domain layer knowledge by the inference layer elements. Coverage checks of the domain layer relations provide feedback that tells us wether all concepts of a domain-type are used in those domain relations that refer to elements of that type. Table 6.1 shows the result of the analysis of the relation *suboptimal-clamping-tool*. The first two paragraphs of the analysis text tell us that the domain concepts ***axial-turning-p*** and ***transverse-turning-p*** are not used in that relation, even though they are potential values for the first argument position of the tuples of that relation (see Table 5.2 for the definition of the relation *suboptimal-clamping-tool*).

The same analysis text (Table 6.1) tells us in the third paragraph that *clamping-jaw* is not mentioned. *Clamping-jaw* is mentioned again in Table 6.2. This table shows the results of analyzing the use of the domain layer knowledge in a role. The analysis tells us that the concept ***clamping-jaw*** can play the role denoted by ***solution-alternative***. This role is the input of the inference actions ***select-ideal*** and ***select-feasible*** (see Figure 5.3). None of the forward chaining rules that represent the implementation of the these inference actions (see Table 5.5) refer to the concept ***clamping-jaw*** in their preconditions. This points to an dead-end in the knowledge base, which should be the topic of a focussed knowledge acquisition session with COKAM.

Table 6.1: The results of an analysis of the relation *suboptimal-clamping-tool.*

```
The domain layer concept AXIAL-TURNING-P is not used in any tuple of the
relation SUBOPTIMAL-CLAMPING-TOOL it should appear in 1. position (called
IMPORTANT-TURNING-REQUIREMENTS)  with the argument description (EXTENSION-*
TURNING-REQUIREMENT).

The domain layer concept TRANSVERSE-TURNING-P is not used in any tuple of the
relation SUBOPTIMAL-CLAMPING-TOOL it should appear in 1. position (called
IMPORTANT-TURNING-REQUIREMENTS)  with the argument description (EXTENSION-*
TURNING-REQUIREMENT).

The domain layer concept CLAMPING-JAW is not used in any tuple of the relation
SUBOPTIMAL-CLAMPING-TOOL it should appear in 0. position (called CLAMPING-TOOL)
with the argument description (INSTANCE CLAMPING-TOOL).
```

Table 6.2: The results of the analysis of the role *instantiated-criterion.*

```
The domain concept CLAMPING-JAW can be assigned to the role SOLUTION-
ALTERNATIVE, but it is not used by any of the inference actions SELECT-IDEAL,
SELECT-FEASIBLE that use SOLUTION-ALTERNATIVE as control- or as input-role.

To change this do AT LEAST ONE of the following:

    Add a tuple to the relation OPTIMAL-CLAMPING-TOOL, which is used by the
    inference action SELECT-IDEAL. CLAMPING-JAW must be mentioned in the 0.
    argument of the tuple, which is of type (INSTANCE CLAMPING-TOOL) and is called
    CLAMPING-TOOL.

    Add a tuple to the relation SUBOPTIMAL-CLAMPING-TOOL, which is used by
    the inference action SELECT-FEASIBLE. CLAMPING-JAW must be mentioned in the 0.
    argument of the tuple, which is of type (INSTANCE CLAMPING-TOOL) and is called
    CLAMPING-TOOL.
```

Table 6.3: The results of the analysis of the inference action *compute-cost.*

```
The condition (Role-Applicable Possible-Solution Clamping-Jaw) derived from
the tuple (PRODUCTION-COST SET-UP-TIME OUR-WORKPIECE (CLAMPING-JAW PRODUCTION-COST
= (+ (<- CLAMPING-JAW :GET 'PRODUCTION-COST) (<- CLAMPING-JAW :GET 'SET-UP-
TIME)))) of the relation PRODUCTION-COST used by the inference action COMPUTE-COST
can never be fired, as none of the inference actions COMPUTE-COST, SELECT-
FEASIBLE, SELECT-IDEAL assign the role POSSIBLE-SOLUTION to the domain concept
CLAMPING-JAW.

To change this do AT LEAST ONE of the following:

    Add a tuple to the relation PRODUCTION-COST, which is used by the
    inference action COMPUTE-COST. CLAMPING-JAW must be mentioned in the argument
    nbr. 2, which is of type (EXTENSION CLAMPING-TOOL).

    Add a tuple to the relation SUBOPTIMAL-CLAMPING-TOOL, which is used by
    the inference action SELECT-FEASIBLE. CLAMPING-JAW must be mentioned in the
    argument nbr. 0, which is of type (INSTANCE CLAMPING-TOOL) and is called
    CLAMPING-TOOL.

    Add a tuple to the relation OPTIMAL-CLAMPING-TOOL, which is used by the
    inference action SELECT-IDEAL. CLAMPING-JAW must be mentioned in the argument
    nbr. 0, which is of type (INSTANCE CLAMPING-TOOL) and is called CLAMPING-TOOL.
```

The analysis of the inference actions produces similar results (see Table 6.3). The inference action *compute-cost* uses the domain relation *production-cost*. The rule-schema that the compiler produces to evaluate the inference action (see Table 5.5) will contain a precondition testing wether *clamping-jaw* currently plays the role *possible-solution*. The knowledge base analyzer determines that this can never be the case, as none of the inference actions that use *possible-solution* as output-role assign it to that concept. The analyzer proposes several ways to remedy this. This information, coined in the terms of the operational model can easily be traced back to the KADS conceptual model, as the operational model is a structure-preserving implementation of the conceptual model.

6.2. Using the Feedback in the Next Session with COKAM

The results of the OMOS analysis, which indicate gaps in the so far acquired knowledge, are used in various ways in the next session with COKAM. In order to check whether the missing knowledge was overlooked when extracting knowledge units from the text, the text can be searched for the respective keywords (e.g., *axial-turning* or *clamping-jaw*). Since COKAM records which text passages contributed to the informal knowledge base, it can be easily determined whether the keywords occur in text passages which contain potentially novel information. In this case the expert has to judge whether these text segments contain information which should be added to the informal knowledge base.

If the uncovered knowledge gaps can not be filled from the initially selected text, the expert is asked for additional literature which covers these topics. The knowledge gaps mentioned in Table 6.1 are due to the fact that our initial text dealt mostly with clamping tool selection for outside turning and only contained a few references to inside turning.

Furthermore it may be advisable to select a new case for which the missing knowledge is supposedly needed and have the expert generate an explanation. This should be done in particular when the knowledge gaps cannot be easily filled from the initial or the additional text or when the correctness of the constructed conceptual model is in question and must be reexamined. Since the OMOS analysis indicates that the knowledge concerning clamping jaws is fragmentary, a case in which a clamping jaw is applied to fix a workpiece should be explained by the expert. If the obtained explanation structure substantially differs from the other explanation structures then this may indicate that the conceptual model must be modified. For instance, the original conceptual model may be too simple, since the relevance of certain criteria only becomes obvious, when cases are encountered for which these criteria cannot be neglected.

7. Summary & Outlook

The combination of COKAM, KADS and OMOS is an experiment that we started to analyze the interaction of different contributions to knowledge acquisition. We spanned the bridge from natural language texts and experts' explanations of previously solved cases (COKAM) via systematic models of problem solving (KADS) to an operational system (OMOS). In these transformation processes we focused on the following questions:

- how can a KADS model be built from explanation structures, and how does the KADS model influence the generation of explanations?
- how can we formalise the KADS model, and what does the formalization tell us about the model?
- what does the formal model propose for the next knowledge acquisition session?

In the present paper we used a toy version of a real world problem to analyze and illustrate our approach. The future work will concentrate on full-blown versions of the process planning problems for rotational parts.

References

[Ak90] Akkermans, H., Balder, J., van Harmelen, F., Schreiber, G., and Wielinga, B. *Formal specifications of knowledge level models* . Tech. Rept. P3178, Task 1.4, Esprit Basic Research Project , REFLECT Repeort, 1990.

[Br87] Breuker, J., Wielinga, B., van Someren, M., de Hoog, R., Schreiber, G., de Greef, P., Bredeweg, B., Wielemaker, J., Billaut, J.P., Davoodi, M., and Hayward, S. *Model-driven knowledge acquisition: Interpretation models.* Tech. Rept. Esprit Project 1098, Deliverable Task A1, University of Amsterdam, Social Sciences Informatics, Amsterdam, 1987.

[Br89] Breuker, J. and Wielinga, B. Models of expertise in knowledge acquisition. In *Topics in Expert System Design, Methodologies and Tools.* North-Holland, Guida, G. and Tasso, C., pp. 265 - 295, Amsterdam, 1989.

[Ch86] Chandrasekaran, B. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert* (Fall 1986), 32 - 30.

[G87] Güsgen, H.W., Junker, U., and Voß, A. Constraints in a hybrid knowledge representation system. In *Proceedings of IJCAI87,* McDermott, J., IJCAI, Morgan Kaufmann, Los Altos, 1987, pp. 30 - 33.

[Ge90] Gertzen, H. *Entscheidungen bei sequenzierter Informationsdarbietung am Bildschirm,* Münster; New-York: Waxmann (1990).

[Hu82] Huber, O. *Entscheiden als Problemlösen,* Bern: Huber (1982).

[Li90] Linster, M. *Declarative problem solving procedures as a basis for knowledge acquisition: A first proposal.* Tech. Rept. 448, Arbeitspapiere der GMD, GMD, St. Augustin, June, 1990.

[Li91] Linster, M. and Musen, M. *Use of the KADS knowledge acquisition method to model the ONCOCIN task of cancer-chemotherapy administration.* 1991, submitted.

[Mc88] McDermott, J. Preliminary steps toward a taxonomy of problem-solving methods. In *Automating Knowledge Acquisition for Expert Systems.* Kluwer Academic, Marcus, S., Ch. 8, pp. 225 - 256, Boston, 1988.

[Mu89] Musen, M.A. *Automated generation of model-based knowledge acquisition tools,* Pitman Publishing, London , Research Notes in Artificial Intelligence(1989).

[Pr87] di Primio, F. and Wittur, K. BABYLON, a meta-interpretation model for handling mixed knowledge representations. In *Proceedings of the 7th International Workshop on Expert Systems and Their Applications,* Avignon, 1987, pp. 821 - 833.

[Ri89] Richter, M.M., Boley, H., Wetter, T., and Warnecke, G. *ARC-TEC: Acquisition, representation and compilation of technical knowledge.* Tech. Rept. Projektantrag, German Research Center for Artificial Intelligence, Postfach 2080,D-6750 Kaiserslautern, 1989.

[Sc88] Schreiber, G., Breuker, J., Bredeweg, B., and Wielinga, B. Modelling in KBS Developement. In *Proceedings of EKAW88,* Boose, J.H., Gaines, B.R., and Linster, M., GMD, St. Augustin, 1988, pp. 7/1 - 7/15.

[Sc90a] Schmidt, G. and Schmalhofer, F. Case-oriented knowledge acquisition from texts. In *Current Trends in Knowledge Acquisition*, Wielinga, B., Boose, J., Gaines, B., Schreiber, G., and van Someren, M., IOS Press, May 1990, pp. 302-312.

[Sc90b] Schmalhofer, F., Kühn, O., and Schmidt, G. *Integrated knowledge acquisition from text, previously solved cases, and expert memories* . Tech. Rept. RR-90-14, Research Report, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), also presented in Proceedings of AAAI Workshop: Knowledge Acquisition for Knowledge Based Systems (1990), December, 1990.

[Sc90c] Schreiber, G., Akkermans, H., and Wielinga, B. On problems with the knowledge level perspective. In *In Proceedings of the 5th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November 1990, pp. 30-1 - 30-14.

[Vo90] Voß, A., Karbach, W., Drouven, U., Lorek, D., and Schuckey, R. *Operationalization of a synthetic problem.* Tech. Rept. 1.2.1, Esprit Basic Research Project P3178 REFLECT, GMD, St. Augustin, April, 1990.

[We] Werkzeuge, S.K.F., Examples for Application (Turning and Milling)Feldmühle AG, Produktbereich SKP-Werkzeuge, D-7333 Ebersbach a.d. Fils, Gottlieb-Häflel-Str. 7, .

[We90] Wetter, T. First Order Logic Foundation of the KADS Conceptual Model. In *Current Trends in Knowledge Acquisition*, Wielinga, B., Boose, J., Gaines, B., Schreiber, G., and van Someren, M., IOS Press, , May 1990, pp. 356-375.

[Wi89] Wielinga, B., Akkermans, H., Schreiber, G., and Balder, J. A knowledge acquisition perspective on knowledge-level models. In *Proceedings of the 4TH KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP*, October 1-6 1989, pp. 36-1-36-22.

From Real World Problems to Domain Characterisations

R. C. Paton, H. S. Nwana,
M. J. R. Shave, T. J. M. Bench-Capon.

The MEKAS Project
Department of Computer Science
University of Liverpool
Liverpool, L69 3BX
England.

rcp@and.cs.liv.ac.uk

Abstract

Several problems associated with current approaches to knowledge acquisition have been identified by various workers. These include the lack of an explicit theoretical basis to many existing methodologies and the requirement to be able to integrate the analysis, elicitation and representation of knowledge into a coherent domain model. The purpose of this paper is to address some of these issues. Taking knowledge acquisition itself to be primarily a modelling concern, we examine three general kinds of model of this process. Following this we describe an explicit theoretical framework for one of these model types, namely, the characterisation of problem solving domains. The practical application of these ideas are found within an approach called SAAGS which seeks to provide a characterisation of a domain. This approach has and is being applied in a number of applications, and we provide some details of how it works using selected results from an investigation of the industrial application of neural networks. The final section of the paper looks at the scope of the SAAGS approach and suggests how it could complement other knowledge acquisition techniques.

1 Introduction

Knowledge acquisition usually involves eliciting, analysing and interpreting the knowledge that an expert uses when solving problems and then transforming this knowledge into a suitable machine representation [KD87]. It is a key stage in the development of knowledge based systems although it continues to present a major hurdle to their construction. The purpose of this paper is to show how a fuller appreciation of the nature of a domain and its relationship to the real world can assist a knowledge engineer in knowledge acquisition. The need for this kind of activity becomes clear when it is appreciated that knowledge acquisition is primarily a modelling activity [BW87, CY89]. The models must be grounded or contextualised in human social and cognitive processes [GS89] as well as in the real world [CY89]. We

have proposed the need for a preliminary knowledge analysis or domain characterisation phase which provides this context or grounding [PN90]. The need for this kind of analysis has emerged from a variety of industrially-sponsored knowledge acquisition projects based at the University of Liverpool e.g. [FH89, HS86, WS89] and is further supported by the requirements of two major industrial collaborators. Some of the major problems associated with knowledge acquisition which we seek to address as we go about modelling domains include the need for:

- Theories of the process necessary to underpin a methodological approach. These should have their philosophical and cognitive foundations stated more explicitly [BW89];
- Current approaches to KBS development to become more structured so that knowledge engineers, who often move too quickly to design and implementation, take account of an adequate cognitive definition and understanding of the domain [SW89];
- Guiding principles which allow a knowledge engineer to navigate a domain and make sense of the mass of information obtained [OM89];
- A means by which an expert and knowledge engineer can harmonise their mental models [RP87]; especially as they seek to communicate with each other and talk about the same things in the real world.
- Knowledge acquisition in its early stages to focus on the domain, rather than be driven by tools which are in turn driven by implementation concerns [WD90, PN90].
- A way to match elicitation techniques to the nature of a domain. Despite numerous tools [see BS89], a major obstacle is that little guidance is available in this area [KB89].

The paper now continues with two sections which address an approach to domain analysis more fully. We then apply the model to a real world problem domain: industrial applications of artificial neural networks. In so doing, we restrict the discussion to selective details concerning the analysis of the domain and because of space limitations do not include details of intermediate outputs (further details can be found in [NP90 & NP90a]).

2 Models of Domains

One of the reasons that there are problems associated with knowledge acquisition is that domains are not only complex (a major issue on its own), but that there can be multiple models of the same domain as it exists in the real world. For example, Brian Gaines noted at EKAW '90 that a need exists to address the modelling issue in relation to the expert and the many kinds of formal and informal models that are produced in a KBS development cycle. We argue that a fuller understanding of the domain itself is

also required. This issue is not restricted to research and development concerned with expert systems, workers in the information systems e.g. [MN87, KO90] and deep knowledge base communities e.g. [MA90] also report problems.

An important place to start concerns an ontological issue, namely, what is being modelled. The issue is non-trivial. There can be confusion in this area (e.g., treating a knowledge level model as the same kind of thing as a psychological model of human competence). For the purposes of the present discussion we consider three approaches to knowledge acquisition which we interpret as developing three distinctive kinds of model (for a fuller discussion see [PN90b]). (Note: this idealised classification focuses on the primary emphasis of a model; specific models are often polytypic in nature. Hence, there is no attempt here to categorise specific methodologies).

2.1 Task-Based Models

Solutions to problems are brought about by constructing models, not of what is in an expert's head, but rather of how behaviours are properly combined (engineered) together. Certain key features are summarised as follows:

- The overall emphasis is the *construction* of what can become a computable problem solving model.
- The key cognitive objects are tasks.
- The central focus is Knowledge Level models which solve problems.
- The role of a knowledge engineer should be passive, s/he is the interpreter of problem solving behaviour into a computable design, primarily using pre-defined primitives.

2.2 Concept-Based Models

The models produced in this approach are models of human cognitive structures. This contrasts with the former approach where the models need not reflect the cognitive processes of human problem solvers, only simulcra of their observable behaviours.

- The overall emphasis of this approach is *clarification* of a human cognitive problem solving model.
- The key cognitive objects are concepts constructed "in the head".
- The central focus is experts who solve problems.
- In many ways the role of a knowledge engineer should be passive, s/he is the interpreter of expert conceptual structures.

2.3 Domain-Based Models

Models constructed in this approach are of both the real world domain, the expert's understanding of it and the emergent product of interaction between the expert, knowledge engineer and the domain itself. Knowledge is not just in people's heads,

there is an environmental embedding in which it exists [GS89, MP91]. For this reason, the initial stages of knowledge acquisition focus on the real world domain and the cognitive domain (models) which people use. In our approach, a knowledge engineer attempts to come to share a similar epistemic horizon on a problem domain as an expert e.g. [RP87]. We describe this as a hermeneutic approach [PN91b]:

- The overall emphasis is the *characterisation* of the domain.
- The key cognitive objects are concepts which have a referential basis in the real world.
- The central foci are domains in the real world and expert understanding which brings about the solution to problems in them.
- In many ways the role of a knowledge engineer should be active, s/he is the interpreter of someone else's epistemic horizon on "what is out there in the real world".

Each of these models emphasises particular aspects of domains and each is valuable to knowledge acquisition for different purposes (related to the emphases). The approach we describe in this paper is the latter type with its emphasis on domain characterisation.

3 Foundations of a Structured Approach to Domain Characterisation

The description of the nature of models developed here has a multidisciplinary basis and uses ideas from the philosophy of science and the cognitive and social sciences. Specifically, models involved in the construction, maintenance and evolution of a domain depend on theories for their existence. Such models, and the theories which form them, have both a cognitive and social nature. This is based on a realist understanding of science as described by Harré [HR70, HR86, HR90] and elaborated in [PN90a]. The argument now developed can be used to account for models of domains and can also be used to describe models of knowledge acquisition. We restrict it to the former category in this section. Having defined the hermeneutic approach (section 2), we now provide its underpinning (this section) and its application (sections 4 and 5).

The basic assumption of our method is that humans construct models in order to deal with the complexities of the real world and these are formed by abstraction; by simplifying complexity. The philosophical and cognitive justification for this approach is discussed more fully in [PN90a]. The real world, that is, that which is referenced, has observable and non-observable features. In the simplest case a model is produced by abstracting from what is observable alone (left hand side of Figure 1). This kind of model will lack explanatory power because it cannot account for the causal relations between its parts; they can be no more than statistical associations.

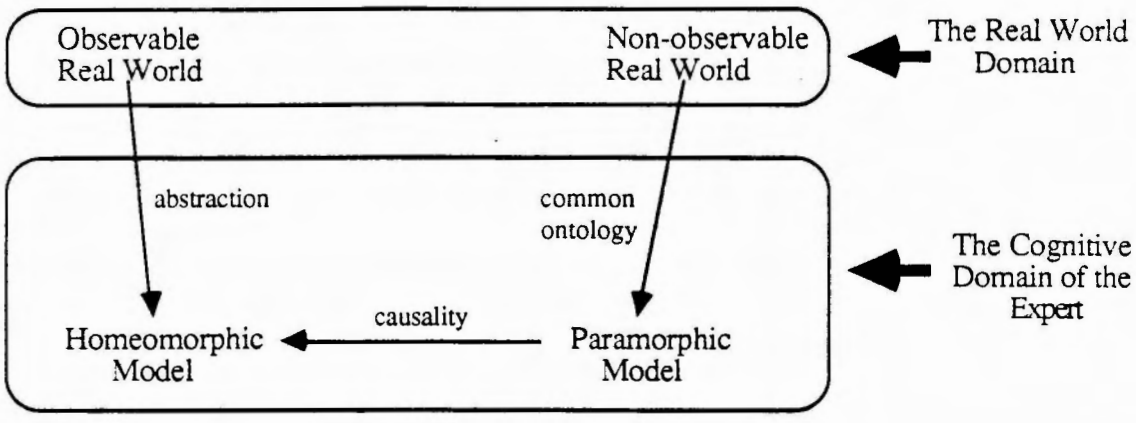


Figure 1 - A Summary Scheme of Model Production (adapted from [HR70])

Most scientific models have explanatory power. This is because the descriptive or homeomorphic model (see Figure 1) is not only formed by abstraction from observables but is also dependent on models of what cannot be observed. These explanatory (paramorphic) models provide the causal framework necessary for explanation. Domains which exhibit explanatory capabilities will have relational or iconic properties (due to their causal infrastructure). If an expert can provide explanations we can anticipate a theory will have iconic (relational) properties [HR86]. This is important when the dynamic aspects of the domain come to be modelled. The basis for paramorphic (explanatory) models is in interpretations of the unobservable real world which share common kinds of entities. It is metaphors which provide the context for such common ontologies [AN84] and this is often noted when ideas are imported from related domains [PN91a]. The role of metaphor and relations with other domains in the construction of models should not not be underestimated. Metaphor provides the context in which analogies and similes are made [SK86]. It has been properly described as the characteristic trope of scientific thought [HR86]. Several knowledge acquisition researchers have reported the importance of understanding metaphor in analysis e.g. [BB89, KL87, PN90] and work continues on its application to A.I. e.g. [AN91, WY91].

It must be emphasised here that the analysis of metaphors developed in our approach is related to global metaphors; that is, common ideas which pervade a domain. In order to illustrate this point, consider a simple investigation concerned with the structure of economic theory. In doing so, we analysed four undergraduate texts for the presence of metaphors which exhibit systemic properties [BM57, HD87, SN80, SH72]. This involved reading the indices of the books and their lists of contents. The product of this analysis, which was non-exhaustive and did not include occasions when 'system' was used on its own, is summarized in Table 1.

Global Metaphor	Associated Properties	Economic Concepts
MACHINE	Balance	of Payments, of Trade, Sterling_
	Elasticity	of Demand, of Supply
	Equilibrium	Equilibrium, Maximum profit_
	Mechanism	Price_, Market_, of Foreign exchange
	Maintenance	Resale price_, Stabilizers, Stability
	Force	Labour_, Market _
	Power	Economic_, Monopoly_, Purchasing_,
	Control	Commodity_, Price_, of Money supply
	Adjustment	of Supply to demand
	Efficiency	Efficiency
	Friction	Frictional unemployment
CIRCUIT	Cycle	Trade_, Business_,
	Circulation	of Money, of Income
	Flow	Goods_, Earnings_, of Cost, of Product
	Drain	Gold_
	Pressure	for Higher wages,
	Capacity	Excess_
ORGANISMIC	Order	Economic
	Growth	Economic
	Movement	Galloping inflation
	Specialisation	Specialisation, Division of labour
GAME	Gambling	and Speculation
	Competition	Perfect_, Competitive supply, of groups
	Dominance	Dominance and deterrance (also war metaphors)

Table 1 - Some Global Metaphors and their Associated Properties in the Domain of Economics

The reader will note that concepts associated with each of the global metaphors can appear across domains of knowledge (middle column) and also define domain specific terms (right hand column). It should be noted that a specific instance of a global metaphor (for some domain) need not reference all the associated concepts. (Note: An overview of the use of metaphors in economics can be found in [MC85]). The contents of Table 1 gives an indication of how associated properties will occur across a range of domains. In the words of Aronson [AN84], they reflect the language of a common ontology (see Figure 1). This will be related to language of the domain and be displayed in its history. The result of this kind of analysis does not provide a complete description of a domain. Rather, it indicates the kinds of concepts that are used. This will help determine the nature of the referents in the real world.

The description of the model shown in Figure 1 can now be applied to our understanding of a domain characterisation.

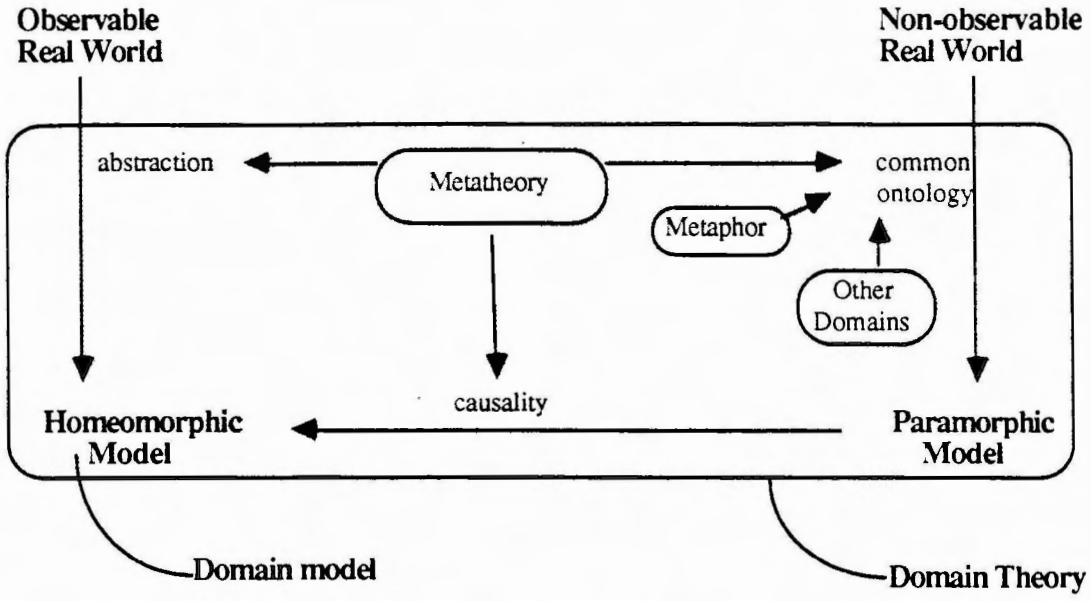


Figure 2 - The Relationship between Model Production and the Top-Level Features of a Domain Characterisation

Figure 2 shows how the domain model (which is usually described in terms of structure and purpose) is related to the overall domain theory. This latter cognitive construction not only changes in time (has a history), but depends upon metatheory, metaphor and relations with other domains for its description. Thus we identify seven fundamental or top-level characteristics of a domain:

- Theory - the conceptual framework used to construct and maintain a domain.
- Metaphor - language used to describe the domain, especially in global terms.
- Metatheoretical constraints - fundamental concepts such as time and causality which provide ontological concepts.
- Relations with other domains - similar bodies of knowledge.
- History - the evolution of the domain in time.
- Structure - the parts, relations and organization of the domain in relation to a model.
- Purpose - the problems which the domain addresses, in terms of their solution.

4 From Domain to Characterisation: The SAAGS Approach in Practice

We have proposed a methodology for knowledge analysis called SAAGS which exploits the model developed in the previous section [PN90]. This is a four stage iterative cyclical modelling process. It receives a loose specification as input and outputs a more comprehensive specification that provides the characterisation of the domain. The overall goal is to detail the structure and purpose of the domain but in the context of its evolving theory. The outputs of SAAGS can be fed into the subsequent design and implementation stages of KBS development. The principal stages are:

1. SPECIFICATION by the production of a characterisation of the domain. The seven top level features must be accounted for in this stage which must also include relevant models as well as descriptive details such as the epistemic boundaries yielded by the analysis.
2. ANTICIPATION of the nature of the domain. The top level features allow the knowledge engineer to anticipate in breadth. The breadth is related to the model of the nature of a domain described in section 3, including likely metaphors and theoretical frameworks that are expected to be used. Anticipations of these characteristics will help structure and guide the domain-based knowledge acquisition process.
3. ACQUISITION of knowledge. This includes: knowledge elicitation from experts and reviewing textbooks, manuals and other knowledge sources. The outcomes from the anticipation stage are used to structure acquisition in a way which relates to the emerging nature of the domain. As such acquisition is driven by domain-related concerns. (Please note: in the terminology of the SAAGS approach, acquisition has this restricted meaning).
4. GENERATION of models including the synthesis of all outputs from the ACQUISITION stage into a collection of models. These models drive the analysis forward and provide the means for confirming, refuting or elaborating on anticipations.

The goal of SAAGS is to produce a domain specification that is comprehensive and relevant and the cycle continues until the goal is reached.

In order to provide the reader with an adequate description of the SAAGS approach in practice, details from a project concerned with the industrial application of neural networks are now given. This was a five-cycle iteration, four with an expert and one initial iteration from text. The total elapsed interview time was less than four hours. The report about this work that is now produced has had to be selective in its contents and further details can be found in [NP90, NP90a].

4.1 The Initial Iteration

The SAAGS approach requires an initial specification, however loose, of the nature of the domain to be characterised. The specification received from the client was:

> " The domain (to be studied) concerns the application of neural networks to industrial problems (Note: specific to the particular company). The need is to identify what types of neural architectures are most suitable for a particular type of application.
> Note: It is necessary to cover a wide range of problems and a wide range of networks.
> Gaps in knowledge/understanding are to be expected and should be identified. "

The initial iteration is very important. It seeks to establish an initial vocabulary of the domain and also attempts to contextualise the domain for the knowledge engineers. Many knowledge acquisition methods prescribe an initial phase of background reading

or orientation by the knowledge engineers [ME88, BW87]. A distinctive feature of the SAAGS approach is that it not only requires this work to be undertaken in a methodical way but it defines a way of going about this activity. It involves the analysis of a textbook or manual recommended by the expert (when available) and is directly tied to the seven anticipated features of a characterisation. Specifically the contents and index are analysed to reveal:

(i) Global metaphors - these help to structure the domain as a whole (section 3),
(ii) Relations with other domains,
(iii) Language associated with the purpose the theory. This often occurs before or after other concepts (e.g., predict, explain, classify, identify, hypothesise),
(iv) Key concepts related to the structure and purpose of the domain (these can appear as frequently occurring concepts in the index).

The outcome of this analysis is that the knowledge engineer has a broad perspective on the nature of the domain.

Knowledge engineers are not passive, dumb, non-interactive agents. They bring certain assumptions with them when embarking on knowledge acquisition. It is crucially important to make this background information explicit. This is achieved by requiring that initial approaches to the characterisation are broadly based. Clearly, any attempt by the knowledge engineer to reflect on his/her background ideas would be long-winded and difficult. The top-level characterisation can guide the knowledge engineer in making assumptions more clear. Without sufficient breadth (or anticipated breadth) to begin with, the knowledge engineer may miss part of the domain. Furthermore, as Breuker & Wielinga point out [BW87], the interpretation of verbal data is impossible without the knowledge engineer having prior assumptions about the processes and knowledge that underly the data. Early anticipations attempt to contextualise the domain model (i.e. structure and purpose) within the real world and the cognitive domain of the expert.

In the present investigation, analysis of a text [AR88] revealed that the domain is a hybrid of several substantial areas of knowledge. The analysis was restricted to an index analysis because the book is a collection of papers. It suggested that the domain

Concept	**Related domains**	**Possible Systemic Metaphors**
Activation	Neurobiology	Circuit, Society
Algorithm	Mathematics	Machine, Text
Association	Psychology	Circuit, Society
Brain	Neurobiology	Circuit, Machine, Organism
Concept	Psychology	Circuit, Text
Feedback	Electronics/Cybernetics	Circuit, Machine
Information	Cybernetics/Computing	Circuit, Machine
Noise	Electronics/Physics	Circuit, Machine
Signal	Electronics	Circuit, Machine

Table 2 - Selection of Concepts (from 56 words extracted)

could make use of a variety of concepts and metaphors some of which are given in Table 2. The reader will note the reference to systemic metaphors in the right hand column of Table 2. "System" is a very common word used in everyday and scientific language. Its epistemological status can be very useful when it comes to characterising a domain. The SAAGS approach has identified the importance of systems thinking in complex domains and seeks to capture the richness of this concept in its analytical procedures. Systemic metaphors are characterised by a set of basic properties (called systemic M-properties [see PN91b]):

(i) interacting parts,
(ii) organisation,
(iii) collective behaviour and whole system functionality.

In order to provide the widest discrimination between systems, the systemic M-properties noted above are associated with a set of systemic metaphors which inherit all the properties given above and include machine, organism, society, circuit, game, text and culture. There are further properties which typify particular systemic metaphors in the language used when talking about them, for example:

Circuit:	flow, transfer, conduit, cycle, transferred thing(s), drain.
Machine:	efficiency, input, process, output, goal, purpose, power.
Organism:	growth, organised complexity, level, adaptability, openness.
Text:	interpretation, context, translation, meaning, style,
Culture:	tradition, expression, ritual, belief, symbol, idea.

The M-properties of the systemic metaphors listed above, though not exclusive to a particular type, are associated most clearly with that type. For example, in saying that a culture is adaptable we are likely (though not necessarily) referencing the organismic metaphor (i.e., culture is like an organism). The caveat for a knowledge engineer is that metaphorical analysis provides possible confirmation of a hypothesis about particular language. It is an approximate technique. Its advantages are that it relates the semantic richness of language to domain models of the real world in a meaningful way.

In addition to the related domain analysis (see above), a more detailed analysis of mathematical concepts was carried out in order to further elaborate the hybrid nature of the domain (see Table 3).

Mathematical Concept	**Sub-domain of Mathematics**
Net	Graph theory
Attractor	Algebra, Chaos
Bayesian	Statistics
Binary code	Crptography, Computer Science
Plane	Geometry, Topology
Matrix	Algebra
Credit assignment problem	Operational Research
Fourier analyser	Analysis

Table 3 - Some Mathematical Concepts (of 71 words extracted)

This preliminary broad perspective allowed us to anticipate that concepts used by the expert could have been imported from:

> Statistics, a range of mathematical sub-domains, neurobiology, psychology, operations research, physics, electronics, general biology, computer science and cybernetics.

The potential importation of concepts and problems from related domains provides a means of anticipating the kinds of global metaphors likely to be used as well as pointing to the deeper theoretical nature of the domain. For example, in the present study, it was anticipated that machine, circuit and organismic metaphorical contexts were likely to be found.

All the information obtained (of which the above is a subset) could now be used to guide the analysis and structure the elicitation sessions. As pointed out elsewhere [PN90], the anticipated nature of a domain is not used to pre-define its nature. The reification of the anticipations is avoided by beginning with an initial loosely-structured, *grand tour* elicitation session [LF87].

4.2 Early Intermediate Iterations

The goal of the first elicitation session was to establish a working rapport with him and to obtain a general appreciation of the epistemic boundaries of the domain through comparison with the anticipated nature following acquisition from text. In order to achieve this, questions were general and quite *grand tour*. The expert was asked about what he did, what the domain is about, if it is related to other domains and to provide some details of the domain in the particular context of application. It is important to note here how anticipations have already guided the knowledge engineer to ask the sort of questions which elicit aspects of domains proposed including metaphor, theory, relations to other domains, history and so forth. This session thus uses both structured and unstructured techniques. The audio-tape record is transcribed and analysed. Primary (paper) models are *generated* from the analysis including a vocabulary of discourse, list of global metaphors, concept map of related domains, list of sortal types and properties/attributes and preliminary relations between the latter. At this stage, the anticipated details are compared with the generated models. For example, the related domains cited by the expert did not include electronics and physics although it did include all the others. The differences between them furnish further questions for the next session with the expert.

The second elicitation session was structured by the outputs from the previous cycle and was designed to clarify points made during the first interview.

4.2.1 Systemic Metaphors Associated with the Domain

The expert used the term "system" often in the first interview. As noted in section 4.1, this is an often used word which, if adequately analysed, can yield a lot of detail about the interrelations, organisation and functionality of the domain.

In order to utilise this situation, a simple pencil-and-paper technique which can be used to probe an expert's understanding has been developed. It is based on an appreciation of the nature of systems, from a scientific viewpoint e.g. [KR85] and a cognitive viewpoint [PA88] and consists of a set of questions and iconic representations which are selected at the anticipation stage of an iterative cycle according to the emerging domain characterisation. Clearly, the selection criteria are important and are governed by the nature of the domain and the ammenability of the expert.

A simple question can provide a valuable ontological probe. It was noted above that the generation stage after the first interview lists the sortal types. In this case the main ones were:

```
Problem
Network
Learning method
Solution
Yourself [i.e. expert]
Client
```

The question was asked:

```
" When you used the term 'system', to which of the items [above] did you refer ? "
```

In this case the expert identified network and learning method. Once this system boundary is established it is possible to cross-check future usages of system [thus establishing a common meaning], and also probe the expert further for his understanding of the system idea. One probe that was used in the present study was:

```
" Systems have certain properties [given below]. Which apply to the system or
systems in the present study ?
        Interrelated parts:
        Levels of organisation:
        Complexity:
        Collective behaviour:
        Purpose:
        Any others ? [The expert gave 'structure of the processing units']
```

The expert noted that all the properties listed applied and included the structure of the processing units. This list is related to the systemic M-properties given in section 4.1. One outcome of the analysis at this stage was the importance placed on the functionality of an individual processing unit and how it contributes to the adaptive nature of the network.

The expert was also given a list of systemic metaphors (see above) and asked if an artificial neural network could be described by each in terms of component parts. This provided structural details about the network:

```
Machine       " can be divided into units and connections ",
Circuit       " can be considered as flow of activity ",
Organism      " not an organism but an integral part thereof ",
Society       " society of interacting individuals...".
Not applicable was given for culture, text and game.
```

This provided the possibility to probe the expert to investigate the dominance of machine language in this domain and establish. However, in the light of the expert's preferences and arguments, it was decided not to pursue this analysis further.

4.2.2 Perspectives on Unsupervised Learning

We look at unsupervised learning here because of the distinctive way the expert used metaphors and how these metaphors were used to demonstrate the functionality of the networks. The concept of unsupervised learning is quite difficult. The expert used metaphors to facilitate understanding. This should be expected, especially when a knowledge engineer is fulfilling the role of a student [BW87] and the expert is the teacher. Thus, experts often communicate what they know in terms of what they think the knowledge engineer will understand. The expert may intentionally manufacture a metaphor in order to promote understanding by the analyst. The analyst must be able to understand these metaphors and the referents they address.

In the first interview the expert sought to differentiate between supervised and unsupervised learning. In order to do so he introduced a simile: a network is like the neural substrate of the brain's cortex (we will not elaborate on the context of the simile here, but it is related to the expert's underlying theory of the domain as discussed in section 4.3 (i)). He talked of the neural substrate in terms of a baby and the trainer as the mother.

```
"In the case of supervised learning the baby says 'dog' and the mother says no,
that is a donkey. In the case of unsupervised learning, the mother does not say
anything."
```

Clearly this provided a way of differentiating between the two kinds of learning and the training role of the mother was also utilised to anticipate properties associated with a trainer in the next iteration.

Consider the following example from the second interview dealing with local feedback for a kind of unsupervised learning.

```
" ...the analogy one can draw is imagine you've got a number of channels and
imagine water running down them. If it only gets into one channel its going to
erode and therefore is going to be deeper and more likely to catch the water the
next time...we found that when water went down the hillside just little
abnormalities might start off a little rivulet. The next time the water went down
it gets deeper and it gets deeper."
```

Here, the expert is seeking to explain the problem by referring to a situation he anticipates the analyst will understand. For the purposes of analysis, the analogy has two contextual dimensions, the circuit metaphor and landscape metaphor. These metaphors provide details about feedback and analysis of the response revealed a theoretical relation between global and local feedback mechanisms and unsupervised learning. In some situations it would have been very useful to probe the expert further to reveal deeper ideas associated with the common ontology (e.g., What are the circuit and landscape analogues ?).

It should be noted that in each of the cases discussed, there is more to the metaphor than its didactic dimension. There are ontological implications [PN91a].

4.2.3 Network Architecture

The first interview with the expert revealed his preference for the idea of architecture as a way of dealing with the dynamic operation of a network. It was clear at the generation stage of the first iteration that network architecture is an important network property. This structured the second elicitation session which initially focussed on the issue.

The expert noted three properties of a network: dimensionality, configuration and "architecture". These were partly described by referring to other network properties. (Note: "architecture" is in quotes because the expert thought it was too static a term to describe the organizational dynamics of the network). However, he did not replace it with another word. From the understanding of the systemic properties associated with architecture we could suggest "organisation" [PN91a]. This kind of response indicates that there is a theoretical structure which is used to inter-relate referents (see Figure 3). The dependency and affective relations provide a degree of explanation about how the behaviour of the network may be described. This explanatory capacity of the verb relation "affects" suggested a causal link between concepts. It is important to distinguish here between the observables and their theoretical inter-relations. For example, that the "architecture" depends on the structure of the unit is based on a theory of how the unit works. This theory was not expressed simply in terms of transfer function but also had a descriptive nature. For example, the threshold function had qualitative ideas associated with it such as "fudge factor". Furthermore, "architecture", dimensionality and configuration are not strictly observables. For example, the former concept references certain aesthetico-mathematical ideas such as harmony, symmetry and order [ER89]. These ideas form a part of the common ontology. The concept map that is produced in Figure 3 is closely related to the text of the transcript. The semantic relations between concepts indicate an underlying theory which can be probed by the knowledge engineer.

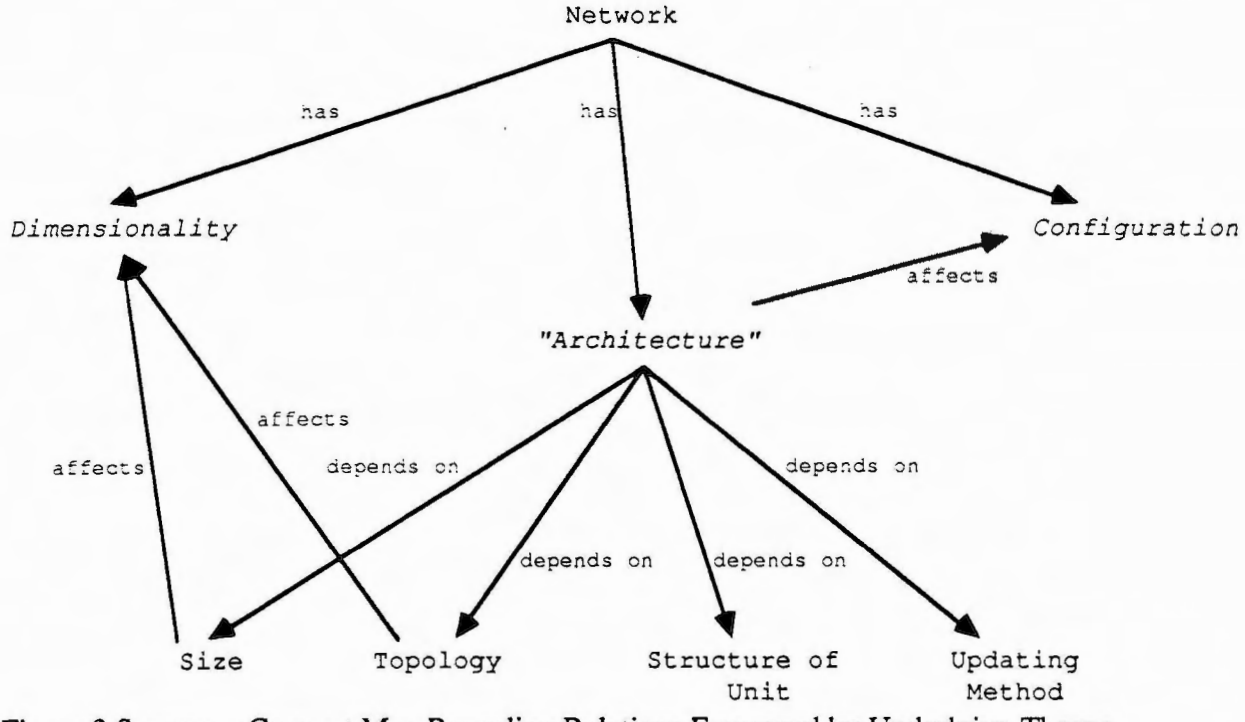


Figure 3 Summary Concept Map Revealing Relations Expressed by Underlying Theory
(Note: concepts in *italics* are abstractions on the other terms. The semantics of the abstraction are expressed by arcs other than "has").

4.3 Later Intermediate Iterations

4.3.1 The Theory of the Domain

The second elicitation session provided a lot of depth of information concerned with network units, architecture, functionality of layered networks and the application to various industrial problems. However, from the point of view of domain characterisation, the issue was: had sufficient detail been obtained ? Was there sufficient breadth to the characterisation? The conclusion made at this stage was no. So far the evidence of a theory was implicit within certain explanations given by the expert (e.g., example (iii) above). It was necessary to acquire more information from the expert and attempt to get more explicit details about his theory of the domain. This is a very important point to make about the SAAGS approach. Decisions about breadth are made in part according to our model of the nature of domains (see section 3), we are not solely dependent on the predilections of the expert or the limitations of an elicitation technique.This can be helpful when an elicitor becomes stuck for questions or has an impression that there is more knowledge available but it is not known how to obtain it.

As a result of the analysis that the characterisation remained incomplete, part of the subsequent elicitation session was structured as a focussed interview in which certain questions were framed which would probe the expert further concerning the theory of the domain. Outputs from the first iteration with the expert recorded that he had experience of expert systems. This provided us with a initial probe for the next interview (based on the top level feature of relations with other domains). It is

important to note why the following question was asked; we returned to the top level to test for breadth (i.e, epistemic boundaries) of the domain. The expert was asked:

" Imagine you are given a problem which can be solved with either a neural network or a rule-based system. Would you make a choice one way or the other ?

The response to this focussed but open-ended probe pointed to issues related to the non-explanatory nature of neural networks and that they are:

"valuable when you are dealing with animal functions, good at mimicking sub-conscious functions ".

As the interview progressed the expert was asked what the theory of the domain was like. The following response he gave provided us with the basic underpinning theory of the domain (for him). Clearly, this does not present a detailed conceptual framework, but it does provide a foundational statement on which a lot of prior and subsequent analysis could be based. It succinctly accounted for his preference for calling neurons "processing units", but much more than this, it summarised his overall perspective on the domain in relation to neurobiology and the machine properties of artificial neural networks:

" There are two different attitudes to the neurophysiology analogy...one is that birds, insects and bats fly. But when the Wright brothers designed their first aeroplane the wings did not flap and it flew. The moral of the story is you don't have to do things the way the biological systems do it in order to be efficient. And the other attitude is that non-biological systems work very poorly. If you can't do something by using a physical system, you can mimick it, i.e, the biological system. I tend to lean to the second view and that's entirely a personal preference, its a question of background. My background is in physiology rather than physics."

This statement underpins a lot of the the structural and functional details of the domain and accounts for his use of the baby/mother metaphor, non-conscious behaviour and even very specific ideas such as the network having a "fit". Could this kind of information have been acquired by other means ? The value of the SAAGS approach is that it requires that this kind of information is acquired. How does it assist in the model of the domain ? The value of such information is that it provides a deeper understanding of the domain and also points to the analogies the expert may use when solving problems in the domain such as seeking to apply particular architectures to novel situations. The discussion above provides the context and depth of characterisation for a rule which the expert provided for deciding on the application of an artificial neural network to a problem:

IF	It is difficult to elicit knowledge from the expert because expert cannot tell you the rules, for example it is too basic (eg. smell; what are the relevant rules to smell or see?)
OR IF	rules are extremely many to be tractable (ie complex)
OR IF	the problem is non-linear
THEN	Use Artificial Neural Networks Approach.

The contextualisation of this rule is very important if a knowledge base is to have explanatory depth. The three conditions are all related to properties of neurobiological systems.

4.3.2 Network Architecture and Problem Classification

As the cycle continued to iterate it became very clear that the large number of particular problem applications in this domain could be classified in terms of network architecture (the expert considered the latter to include network topology and the updating mechanism, see section 4.2 (iii)). The nature of the emerging analysis suggested that a card sort technique [BS87] would clarify the expert's constructs in this area and a subsequent laddered grid analysis was selected to provide the hierarchical relationships amongst the problems. The important point to note here is that the selection of this technique was governed by the emerging nature of the domain, that is, there were very good reasons for making such a choice. Some of the results of this analysis are shown in Table 4.

Network Topology	Learning Type	Suitable Problems	Example
Hopfield	Associative	Combinatoric Optimisation Problems	Travelling Salesman (link between A & B is connection; weight between A & B is distance)
2-layered nets	Supervised/ Unsupervised	Linear/Convex Problems	
Multi-layered nets	Supervised	Pattern Recognition	EX-OR
	Unsupervised	Feature Extraction Cluster Analysis Classification Identification Prediction	

Table 4 - Relations between Network Architecture and Problem Classification

5 Conclusions & Discussion: The Scope of the SAAGS Approach

In this paper, we have argued the case for domain characterisation, stressing the point that some of the major obstacles to the latter lie, to a certain degree, in the lack of adequate attention addressed to this stage 1 activity of Figure 4.

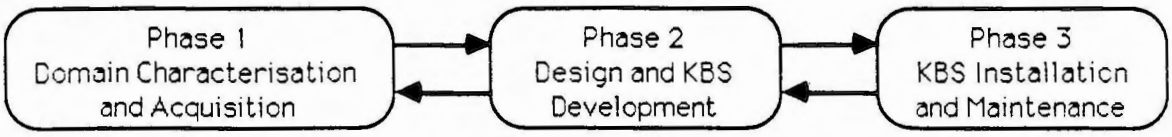


Figure 4 - Phases of KBS Development

After discussing its philosophical foundations, we presented an approach called SAAGS designed to assist domain characterisation. The way it has been carried out in practice has been discussed and a case study involving the domain of neural nets has also been described in some detail; it was not possible to present the full results of our characterisation for lack of space.

The scope of SAAGS has not been mentioned so far but it is crucial. Firstly, it is not necessary to exploit the method only in order to develop a KBS. SAAGS could just as well be used to analyse a domain so as to, say, reveal gaps in the domain and/or investigate the feasibility for developing a KBS. For example, a key goal of the neural networks investigation was to reveal the gaps in the domain. The characterisation of this domain numbered up to twenty pages [NP90].

Consider the representation below which illustrates some of the gaps we identified in this domain as we were required to do in the initial loose specification.

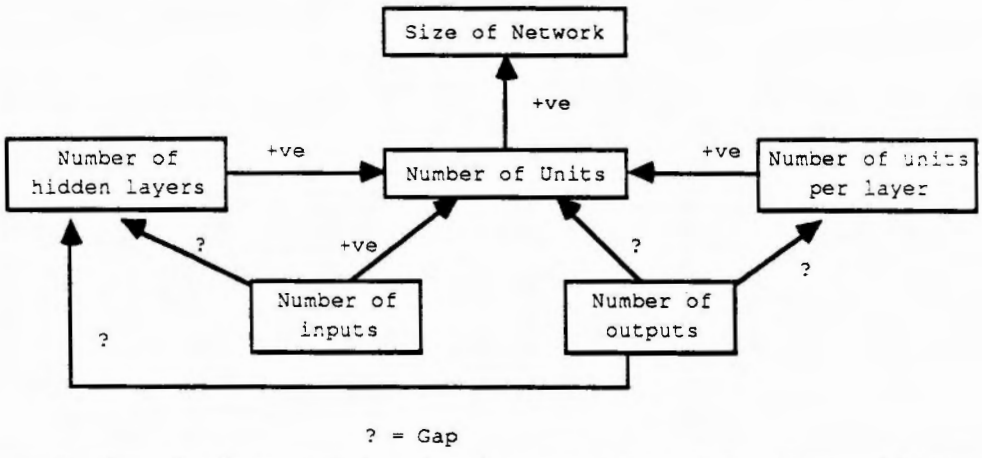


Figure 5 - Part of a Representation showing gaps in knowledge (Layered Nets only)

(Note: the above figure should be read as *number of units* in network has a positive influence on *size of network*; *number of inputs* has a positive influence on number of units; the relationship between *number of inputs* and *number of hidden layers* (if any) is not known and hence constitutes a gap, etc. For example, what is relation between number of inputs and number of hidden layers - does an increase in number of inputs require more hidden layers and by how much? What is the theory behind this (if any)? There are plenty of such questions thrown up by the representation above which are still to be answered. However, there are a few loose heuristics.)

Imagine one was expected to construct a hypothetical expert system for providing advice on issues raised by the above representation e.g. given the number of inputs and outputs to the network, provide advice on the number of units, number of hidden

layers and so forth. It is clear that a KBS could not be constructed for this domain as there are so many gaps. The gaps exist because there is no theory to underpin the relationships: the expert could only provide a loose and unjustified heuristics (e.g., number of units in inner layer = 3/4 * mean (number of inputs + number of outputs)). Thus, one sees how a SAAGS analysis is capable of determining the feasibility for KBS construction. However, imagine all the gaps in Figure 5 were resolved and they were clear theories linking the boxes, then a deep qualitative (and possibly quantitative) model results. This is a important as this suggests SAAGS also yields a deep reasoning model of the domain that obtained from rules. Such a deep model is invaluable for second generation expert systems which combine heuristic reasoning based on rules with deep reasoning [ST86].

The approach also clearly provided us with a way to make sense of the domain, mainly using our seven-featured scheme of characteristics for a domain [PN90]. Furthermore, the products of the characterisation could definitely feed into existing tools/techniques. The glossary it yielded could feed into the lexicon of toolkits like SHELLEY [AW90] or KEATS [ME88]; the static representations of the *structure* and relationships could feed into the domain layer of KADS say; the dynamic outcomes of our method (addressing issues of *purpose* of the domain in terms of their goals and functionality), mostly represented now in some form of causal networks such as Figure 5, clearly reveal the *tasks* ; and some of the *inferences* in the domain may again suggest the usage of certain generic tasks (e.g. in KADS). It has been emphasised that for the expert and the knowledge engineer to merge their horizons, the metaphors/theories' context and usage must be clearly understood; metaphors play a great part in this process as has been emphasised in this paper. Constructs and elements which are yielded by the analysis, would feed easily into state-of-the-art tools like AQUINAS. Perhaps most importantly, this form of characterisation could provide a specification document which could help minimise the costs incurred at later design, development and maintenance phases of KBS development.

In summary, we believe our work contributes to knowledge acquisition by:

- Supporting a deeper theory of knowledge acquisition.
- Suggesting techniques to get the expert and knowledge engineer harmonise their mental models.
- Providing analytical techniques for characterising a domain so as to reveal what elicitation methods are used and when. This can be used to address the representation mismatch problem.
- Providing techniques to navigate and make sense of a domain as the analysis progresses and the amount of knowledge has to be managed as a coherent whole.
- Encouraging the nature of a domain to guide the acquisition process as well as the final form of the knowledge base.

- Providing an adequate characterisation of the domain before entering later phases of acquisition.
- Providing a comprehensive understanding/characterisation of the domain [while revealing gaps in knowledge] as well as 'outputs' which feeds into later tools and techniques of the knowledge engineering cycle.

Acknowledgements
Thanks to our expert Brian Ward, Professor Rom Harré for his advice and comments related to section 3, and Dr. Mary Meyer for her feedback about section 2. Comments on a related paper from Dr Marc Linster and Dr Anjo Anjewierden have also proved useful. The MEKAS Project is sponsored by Shell Research Ltd (Thornton Research Centre) and Unilever Research (Port Sunlight Laboratory).

References

[AN84] Aronson, J. L. (1984), *A Realist Philosophy of Science*, London: Macmillan.

[AN91] Aronson, J.L. (1991, in press), "Verisimilitude and Type Hierarchies", To appear in *Philosophical Topics*.

[AR88] Anderson, J. A. & Rosenfeld, E. (eds), (1988), *Neurocomputing*, London: MIT Press.

[AW90] Anjewierden, A., Wielemaker, J. & Toussaint, C. (1990), "SHELLEY - Computer Aided Knowledge Engineering", In Wielinga, B, Boose, J., Gaines, B., Schreiber, G. & van Someren, M. (eds), *Current Trends in Knowledge Acquisition*, Amsterdam: IOS, 41-60.

[BB89] Brulé, J.F. & Blount, A. (1989),*Knowledge Acquisition*, New York: McGraw-Hill.

[BM57] Benham, F. (1957), *Economics*, 5th Edition London: Pitman.

[BS87] Burton, M. & Shadbolt, N. (1987), "Knowledge Engineering", *Technical Report 87-2-1*, Department of Psychology, The University of Nottingham.

[BW87] Breuker, J. & Wielinga, B. (1987), "Use of Models in the Interpretation of Verbal Data", in Kidd, A. (ed), *Knowledge Acquisition for Expert Systems: A Practical Handbook,* London: Plenum, 17 - 44.

[CY89] Clancey, W. J (1989), "The Knowledge Level Reinterpreted: Modeling How Systems Interact", *Machine Learning* **4**, 285-291.

[ER90] Engler, G. (1990), "Aesthetics in Science and Art", *British Journal of Aesthetics* **30** (1), 24-33.

[FH89] Finch, I. (1989), "Explanation systems in engineering domains", *Proceedings of the International conference on systems in engineering*, Wuhan, China.

[GS89] Gaines, B.R. (1989), "Social and cognitive processes in knowledge acquisition", Knowledge Acquisition, **1**, 39 - 58.

[HD87] Holland, S. (1987), *The Market Economy* , London: Weidenfield and Nicholson.

[HR70] Harré, R. (1970), *The Principles of Scientific Thinking*, London: Macmillan.

[HR86] Harré, R. (1986), *Varieties of Realism*, Oxford: Blackwell.

[HR90] Harré, R. (1990), Personal communication.

[HS86] Hughes, S. (1986), "Question classification in rule-based systems", In Bramer, M. A. (ed), *Research and Development in Expert Systems III*, Cambridge: University Press, 123-131.

[KB87] Kitto, C. M. & Boose, J. H. (1987), "Heuristics for expertise transfer: an implementation of a dialogue manager for knowledge acquisition", *International Journal of Man-Machine Studies* **26**, 183-202.

[KL87] Kornell, J. (1987), "Formal Thought and Narrative Thought in Knowledge Acquisition", *International Journal of Man-Machine Studies*, **26**, 203-212.

[KO90] Kangassalo, H. (1990), "Foundations of Conceptual Modelling: A Theory Construction View", In Kangassalo, H., Ohsuga, S. & Jaakkola, H. (eds), *Information Modelling and Knowledge Bases*, Amsterdam: IOS, 19-35.

[KR85] Klir, G.J. (1985), *Architecture of Systems Problem Solving*, New York: Plenum.

[LF87] LaFrance, M. (1987), "The Knowledge Acquisition Grid: a method for training knowledge engineers", *International Journal of Man-Machine Studies* **26**, 245-255.

[MA90] Morgan, G (1990), "Five years of Deep Knowledge: A Personal View", *AISBQuarterly* **74**, 20-31.

[MC85] McCloskey, D.N. (1985), *The Rhetoric of Economics*, Brighton: Wheatsheaf.

[ME88] Motta, E., Eisenstadt, M., Pitman, K. & West, M. (1988), "Knowledge acquisition in KEATS: The knowledge engineer's assistant", *Expert Systems* , **5**(2).

[MN87] Madsen, K.H. (1987), "Breakthrough by Breakdown: Metaphors in Structured Domains", Information Science Department, Aarhus University, Denmark.

[MP91] Meyer, M.A., Picard, R.R. & Ross, J.M. (1991) "The Ethnological Method of Participant Observation as a Technique for Manual Knowledge Acquisition", Report LAUR-90-1654, Los Alamos National Laboratory, New Mexico, U.S.A.

[NP90] Nwana, H.S. & Paton, R.C. (1990), "Report of a Case Study on the Application of the SAAGS Approach to a Domain Concerned with Neural Networks", MEKAS Report Number 12, The Department of Computer Science, Liverpool University, 15th November, 1990.

[NP90a] Nwana, H.S., Paton, R.C., Shave, M. J. R. & Bench-Capon, T. J. M. (1990), "The Relevance of Domain Characterisation to Knowledge Acquisition for Second Generation Expert Systems", MEKAS Report Number 17, Department of Computer Science, The University of Liverpool, December 19. Submitted for publication.

[OM89] O'Neill, M. & Morris, A. (1989), "Expert systems in the United Kingdom: an evaluation of development methodologies", *Expert Systems* **6**(2), 90-98.

[PA88] Paton, R. C., (1988), *Some Investigations into the Use and Meaning of System in Biological Education*, Unpublished Ph.D. Thesis. The University of Leeds.

[PN90] Paton, R. C. & Nwana, H. S. (1990), "Domain Characterisation through Knowledge Analysis", *Proceedings of AAAI-90 Workshop on Knowledge Acquisition: Practical Tools and Techniques*, Boston, Massachussetts, July.

[PN90a] Paton, R.C., Nwana, H.S., Shave, M.J.R. & Bench-Capon, T.J.M.(1990a), "Foundations of a Structured Approach to Domain Characterisation", MEKAS Report Number 13, Department of Computer Science, The University of Liverpool, November 24. Submitted for publication. This is an extended version of a paper presented at the Eighth Workshop of the European Society for the Study of Cognitive Systems (ESSCS), Oxford. September, 1990.

[PN90b] Paton, R. C., Nwana, H. S., Shave, M. J. R. & Bench-Capon, T. J. M. (1990b), "Towards a Classification of Approaches to Knowledge Acquisition: A Proposal", MEKAS Report Number 15, Department of Computer Science, The University of Liverpool, December 8.

[PN91a] Paton, R. C., Nwana, H. S., Shave, M. J. R., Bench-Capon, T. J. M. & Hughes, S. (1991a) "Transfer of Natural Metaphors to Parallel Problem Solving Applications", in Schwefel, H-P & Maenner, R. (eds), *Parallel Problem Solving from Nature, Lecture Notes in Computer Science*, Berlin: Springer-Verlag, 355-365.

[PN91b] Paton, R. C., Nwana, H. S., Shave, M. J. R. & Bench-Capon, T. J. M. (1991b), "Systemic Metaphor Analysis for Knowledge Based System Development", to appear in Jackson, M.C. et al (eds), *Systems Thinking in Europe*, London: Plenum.

[SK86] Soskice, J.M. (1985), *Metaphor and Religious Language*, Oxford: Clarendon.

[SN80] Samuelson, P.A. (1980), *Economics* , 11th Edition New York: McGraw-Hill.

[ST86] Steels, L. (1986), "Second Generation Expert Systems", In Bramer, M. A. (ed), *Research and Development in Expert Systems III*, Cambridge, 175-183.

[SW89] Shaw, M. L. G. (1989), "A Grid-Based Tool for Knowledge Acquisition", *Proceedings of IJCAI-1989 Workshop on Knowledge Acquisition*, Detroit, Michigan, 19-22.

[WD90] Woodward, B. (1990), "Knowledge acquisition at the front end: defining the domain", *Knowledge Acquisition* **2**, 73-94.

[WS89] Watson, I. D., Shave, M. J. & Moralee, D. S. (1989), "A knowledge analysis methodology using an intermediate knowledge representation based on conceptual graphs", *Proceedings of the 9th Int. Workshop on Expert Systems and their Applications* **1**, Avignon, France, 183-198.

[WY91] Way, E.C. (1991), *Metaphor and Knowledge Representation*, Dordrecht: Kluwer.

Acquiring different types of knowledge: A distributed architecture

D. R. Pugh and C. J. Price

Department of Computer Science
University College of Wales
Aberystwyth
Dyfed
SY23 3BZ
United Kingdom

Abstract

This paper describes work into building flexible knowledge acquisition systems. The eventual aim of this work is that it will be able to adapt to a range of different interpretation tasks (KADS [1]) and hence be part of a complete knowledge acquisition methodology.

The paper contains details of a number of knowledge acquisition modules constructed to match different *generic tasks*, and shows how they can be combined in a way which facilitates the knowledge acquisition for a multitude of applications. Each of these applications is made up from a different combination of generic tasks. An example of this architecture working to perform the knowledge acquisition for a particular application is described.

Finally, the paper compares and contrasts the architecture detailed here with work that has already been accomplished in this field.

1 Introduction

This paper describes a knowledge acquisition program based upon a distributed architecture. It is a program which consists of a number of individual knowledge acquisition modules, each of which has the capability of performing the acquisition of one particular type of knowledge.

The exact nature of these different types of knowledge will be one of the issues discussed in the paper. Examples include classification hierarchies, deep models and pattern matching.

Each knowledge acquisition module is allowed to communicate with the other modules via a simple but effective communication mechanism. The reason for this is that an expert

using the system can then perceive it as a single automated knowledge acquisition program and not as a set of unrelated knowledge acquisition modules. The distributed architecture allows the different knowledge acquisition modules to perform their job of obtaining different types of knowledge in a cooperative manner.

The program is controlled by a management module which oversees the work being carried out by the knowledge acquisition modules. Which module is being used at any one time is however dictated by the requirements of the expert using the program ie. the answers that are given to previous questions.

The paper describes three knowledge acquisition modules and the communication mechanism used to link these modules together to form a coherent system. It is split into the following sections:

Knowledge acquisition and the generic tasks. This section firstly describes the *interaction problem* and goes on to discuss the theory of generic tasks and how they use the interaction problem to their advantage. Finally, the section discusses the implications this has regarding the construction of automated knowledge acquisition modules.

Knowledge acquisition modules. The next four sections describe the knowledge acquisition modules already constructed, and other modules which will be constructed in the future.

Communication between generic tasks. This section shows how generic tasks can communicate with each other and gives an example application constructed from a number of generic tasks.

Communication between knowledge acquisition modules. This section describes the need for a flexible communications mechanism between the knowledge acquisition modules. It then shows how this flexibility is achieved in terms of a general purpose architecture. An example is then given of how the knowledge acquisition program would be used.

Related work. This last section compares and contrasts the knowledge acquisition program with other related programs.

2 Generic tasks

2.1 The interaction problem

When building knowledge-based systems, how the knowledge is represented has a close relationship to how it will be used to solve problems; knowledge is dependent on its use. This has been called the *interaction problem* [2], and has been described as:

Representing knowledge for the purpose of solving some problem is strongly affected by the nature of the problem and by the inference strategy to be applied to the knowledge. [3]

Generic tasks provide a way of using the interaction problem to advantage — instead of having an uniform representation, each generic task has a representation that is apt for the task being performed and a reasoning mechanism which exploits the representation. Being able to choose from a range of generic tasks when building an application allows efficient and effective construction of systems.

2.2 What is a generic task?

Chandrasekaran [4, 5] describes a generic task as an elementary generic combination of a *problem*, *representation* and *inference strategy*.

Problem. The types of information the task takes as input and the information produced as a result of performing the task. This defines the functionality of the task.

Representation. A way to represent and organise the knowledge that is needed to perform the generic task. This includes a vocabulary of knowledge types, i.e. the knowledge constructs that the representation language should have.

Inference strategy. The algorithm, control and problem solving that the task uses. This provides a vocabulary for inference and control of the task.

As the name implies, a generic task is a general solution for a problem; when the task contains knowledge about a particular domain it becomes a specific problem-solver for that domain. The domain knowledge will be stored in the predefined representation of the task, allowing the inferencing mechanism to make use of this knowledge to obtain the required results.

Some examples of generic tasks listed by Chandrasekaran are: hierarchical classification; plan selection and refinement; hypothesis matching. Each of these generic tasks provide the representation and inferencing for a different type of knowledge.

These generic tasks are at a lower level than an application (diagnosis, planning etc.) but are more efficient and explicit in their control strategies than forward and backward chaining rule based systems.

2.3 Generic tasks and knowledge acquisition

The idea behind generic tasks is that they can be combined in any way necessary to produce a problem solver for a particular application. This provides a challenge for automated knowledge acquisition. A knowledge acquisition program for one combination of generic tasks would be useless if that combination changed. A better solution is to produce one knowledge acquisition module for each generic task, then to build the knowledge

acquisition modules into an architecture which allows them to be used in any order, any number of times. Different orderings of generic tasks can be accommodated by calling upon different modules as required. This strategy is not without its problems and these are discussed in section 9.

3 Knowledge acquisition modules

A number of languages have been designed and implemented by Chandrasekaran's group at Ohio State University [5], each of which identify a particular generic task. These include CSRL (Conceptual Structures Representation Language) for the generic task hierarchical classification; RA (Recognition Agents) for pattern matching; IDABLE (Intelligent DAta Base LanguagE), an intelligent database; PEIRCE (named after C.B. Peirce, who first described the form of inferencing known as abduction) for constructing the best explanation for a particular situation from a number of possible hypotheses; and FR (Functional Reasoning) which reasons about a device in terms of its functionality, much of the work on which has been carried out by Sticklen [6].

Three knowledge acquisition modules are in various stages of construction. Each module is associated with one generic task; to date *Triangle* (for CSRL), *Puzzle* (for RA), and *Machine* (for FR) prototypes have been built. These next three sections show that it is indeed possible to construct knowledge acquisition modules for generic tasks.

Triangle will be described in some depth in order to illustrate the principles of the process, and the other two modules will be described briefly.

4 *Triangle*

4.1 Conceptual Structures Representation Language (CSRL)

CSRL has an inferencing mechanism based upon a technique called *establish–refine*. The representation it requires is a tree structure; each node of the tree is a specialisation of its parent node (excluding the root node).

In CSRL each node is called a specialist. Each of the specialists in the hierarchy requires a means of identifying how likely it is to be true. For example, if the specialist was *liver disease*, how likely is it that *liver disease* is true? The answer could range from *very unlikely* to *very likely*. If the value is above a certain value (perhaps *fairly likely* in this case) then it is most probable that the specialist is true and its state can be set to *established*. If not, then the state can either be set to *rejected* or *suspended*. If the state of a specialist is *suspended* and further information arises, this could be used to *establish* or *reject* the specialist.

Once a specialist has *established*, it can then invoke the *refine* command. This gives a call to each of the specialist's sub-specialists asking them to try to *establish* themselves. For each specialist that manages to *establish* itself the *refine* command is in turn given to its sub-specialists. This continues until no specialists can *establish* or the leaf nodes of

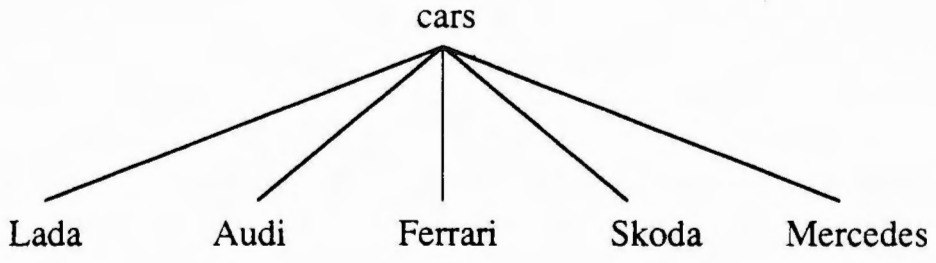


Figure 1: **Parent and leaf nodes**

the hierarchy are reached. The further the specialists *establish* towards the leaf nodes the more information is found out.

4.2 Triangle's knowledge acquisition capabilities

The representation used for CSRL is very much dependent upon the *establish-refine* inferencing mechanism. The data must be stored in the correct representation for the inferencing to work. *Triangle*'s job is to collect the data and put it into the correct representation, thus allowing *establish–refine* to work.

The way in which this representation is built up is as follows:

What elements are we trying to differentiate? This will give the leaf nodes of the classification hierarchy. For example, where choosing between cars the elements might be Lada, Ferrari, Audi, Mercedes and Skoda.

To what group do these elements belong? This should give the name of the root node of the hierarchy. In our example the answer would be "cars". Figure 1 shows the hierarchy so far. It is possible to give more than one root node, for which separate hierarchies would be constructed. If more than one hierarchy is required then *Triangle* finds out which leaf elements belong to which root element.

Can the root nodes be divided further within the confines of their parent node? This involves the more difficult task of finding the nodes between the leaves and the root. Answers in this example might be "expensive" and "cheap". Figure 2 shows the progression in constructing our hierarchy.

Can these internal nodes be grouped within the confines of their parent node? Can the hierarchy be broken down further at the point between these nodes and their parent node? In the example, can "expensive" and "cheap" be grouped further? For each new parent-sibling structure that is created by the addition of internal nodes (boxes A and B in Figure 2) the whole of the above process is applied until the tree can be split no further.

The representation that *Triangle* gives is converted into CSRL code which can then be executed. In order to use CSRL sensibly, other generic tasks must be used to produce

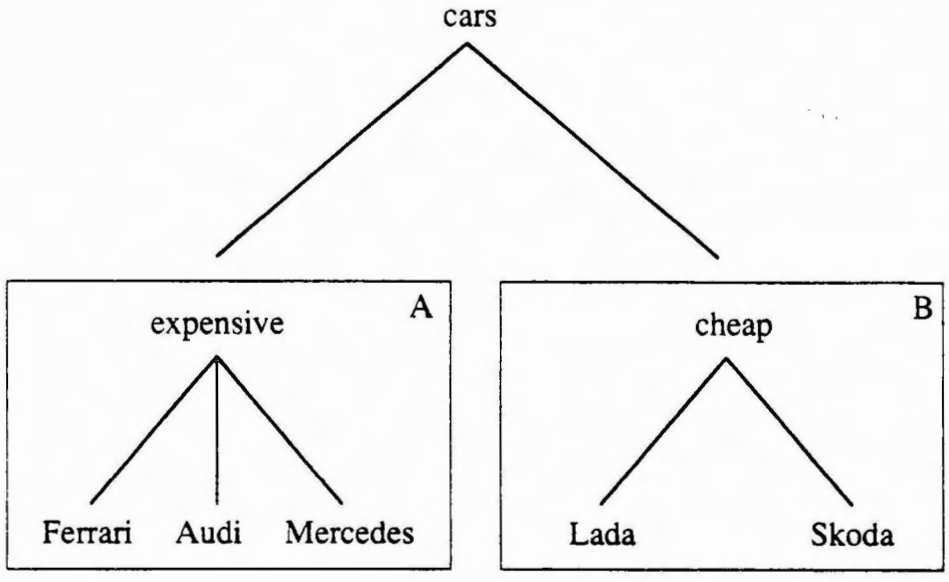


Figure 2: **Parent and leaf nodes**

related information. For example, to decide if a specialist is to *establish*, then a call to RA, FR, IDABLE etc. may be necessary. If a knowledge-based system is to be constructed that will work, then other knowledge acquisition modules will have to perform their job of acquiring knowledge for related tasks.

5 *Puzzle*

5.1 Knowledge acquisition for Recognition Agents (RAs)

Table 1 gives an example Recognition Agent in tabular form. The table consists of three parts. The first gives the possible values that certain variables can take (? means any value); the second specifies how many of these entries must match correctly; and the third shows the result returned by this particular agent. Other information is also required such as what value will be returned if no match is found. In this case it would probably be *unlikely*.

The inferencing used by RA involves going through the table systematically line by line seeing if the actual variable values match the left hand side (taking into account the second column). As soon as a match is found RA returns the result.

The first thing *Puzzle* does is to find out the range of possible results that can be returned. Is it 3 values of *no*, *neutral*, or *yes*; 6 values; or even 9 values ranging from *very unlikely* to *very likely*?

Next *Puzzle* finds out the column headings — the values of what objects are required to decide result. This is followed by asking for the list of allowable values that these columns can take for each of the possible results. Not all possible results require a list of acceptable values for the objects.

Table 1: **RA: Likelihood of car being a Ferrari**

Fast	Well built	Sporty	Italian	Red	*no. of matched attributes*	Result
yes	yes	yes	yes	?	4	*very likely*
yes	yes	yes	yes	?	3	*likely*
yes	yes	?	?	yes	–	*likely*
?	?	yes	yes	yes	–	*likely*
yes	yes	yes	yes	?	2	*somewhat likely*
no	no	?	?	?	–	*very unlikely*

For each of the rows that is initially constructed by *Puzzle*, it is assumed that each of the entries must match for the result to be deemed true. The final step involves asking the expert what the result would be if only one/ two/ three/ etc. of these entries were correct. This allows for possibilities comparable to rows 1, 2 and 5 of Table 1.

These basic steps enable *Puzzle* to construct a complete Recognition Agent. This description has been greatly simplified. Much other information also needs to be gathered, but a full description of the process would be beyond the confines of this paper.

6 *Machine*

6.1 Knowledge acquisition for Functional Reasoning (FR)

A knowledge acquisition module has been constructed for a FR-like generic task. A full description of this module can be found in [7].

The Functional Reasoning generic task allows for the description of a device (or system) in terms of a model of its functionality and the functionality of its individual components. The resulting model is many layered; the further we probe into its sub-components the more information about the device can be obtained.

The model can be used as it is to simulate the device. But more importantly for diagnostic programs, it is possible to change the functionality of part of the device (or one of its components) and then to simulate the device with this changed functionality. This shows the altered functionality of the device given these changes.

Machine allows for the construction of this multi-layered model, building up an outer layer of general functions that the model performs and then taking each sub-component in turn finding out their functionality and in turn the functionality of their sub-components. This is repeated until the expert has gone as "deep" into the model as required.

7 Further work on knowledge acquisition modules

The design of the architecture includes knowledge acquisition modules for five generic tasks at present, two of which (*Genius* (for IDABLE) and *Excuse* (for PEIRCE)) have not been constructed.

IDABLE is an intelligent database which allows access to information like a normal database, but also allows questions such as *Is this a normal value?* and provides a means of abstracting other information from available data.

PEIRCE's role is to give a *best explanation* for occurrences of data from a set of possible hypotheses.

Once these two further knowledge acquisition modules have been constructed, it is hoped that the resulting system will be capable of performing the knowledge acquisition for a range of knowledge-based systems which have a similar structure of generic tasks.

8 Communication between generic tasks

So far, we have discussed the generic task concept, and shown that for each task it is possible to produce a knowledge acquisition module. However, it is not usually the case that one generic task can be used to produce a knowledge-based system. More typically, an application will need to be made up of a number of generic tasks linked to form a coherent system.

This poses two problems. Firstly, how do the generic tasks communicate to produce a problem solver for a particular application? Secondly, how do we perform the knowledge acquisition for such a composite system.

Chandrasekaran's group at Ohio University have devised the *dragon* as a means of communication. This section is dedicated to the description of such dragons and of an example application being produced from a combination of generic tasks. Work has also been carried out by Punch [8] on communication between tasks but that work deals more with the communication between generic tasks and the "outside world", and will not be discussed here.

8.1 The dragon

Each generic task is made up of a number of parts. CSRL has specialists, RA has Recognition Agents, FR has devices/ sub-devices and so on. Each of these individual parts is known as a *dragon.*

Dragons communicate information in much the same way a function can communicate information back to a calling function or procedure in, for example, Pascal or C.

For example, if a CSRL specialist uses the generic task RA to make the choice of whether it can *establish* or not, then the Recognition Agent used will be called by name from within that specialist. The Recognition Agent will return its result to the specialist whose actions will then determined by this result.

This method of calling upon the services and receiving the results from a particular task is common to all of the generic tasks.

8.2 An example application

More information about the following example can be found in [6]. MDX2 is a application that was written by Sticklen to show that generic tasks can be used to build a knowledge-based system; MDX2 is functionally the same as MDX [9] only it uses generic tasks to obtain the goals. The generic task structure used by MDX2 is shown in Figure 3. The arrows refer to connections between the generic tasks and the questions are the requests the tasks are making of one another.

Control of MDX2 is very much in the hands of PEIRCE. It first consults IDABLE to see what needs to be accounted for; and IDABLE returns a number of signs and symptoms. The CSRL classifiers (there are a total of 10 in MDX2) are then called upon to see how much of the data they can account for. Not all classifiers are used. The ones selected depend upon the results of running through a checklist of signs and symptoms that each CSRL classifier is capable of accounting for. Only ones that can account for a reasonable amount of data are invoked; this is done to save on calculation time. The CSRL classifiers employ the services of RA and FR to decide which of their individual specialists are to *establish*. (RA uses compiled pattern matching knowledge and FR simulates a functional model; the more appropriate being chosen in each situation.) Both RA and FR use IDABLE to find particular values of data. The CSRL specialists then return the results to PEIRCE which checks to see if all signs and symptoms have been accounted for. If so, repeated information is removed and the user is given the diagnosis (with an accounting scheme — describing how the diagnosis was decided upon). If not, more information can be requested from the user (by IDABLE) and the same steps are followed through again.

9 Communication between knowledge acquisition modules

If it was possible to allow *Triangle*, *Puzzle* etc. to perform their jobs separately, then, to produce a knowledge-based system, all we would have to do is to combine the results of each of these knowledge acquisition modules. However, using the modules separately forces the expert to think in a way that is not natural; it is not the job of the expert to break down the domain into different types of knowledge.

This implies that an alternative means of performing the knowledge acquisition must be found, one which is flexible enough to perform the knowledge acquisition for any combination of generic tasks and yet specific enough that the program can be run as just one knowledge acquisition program.

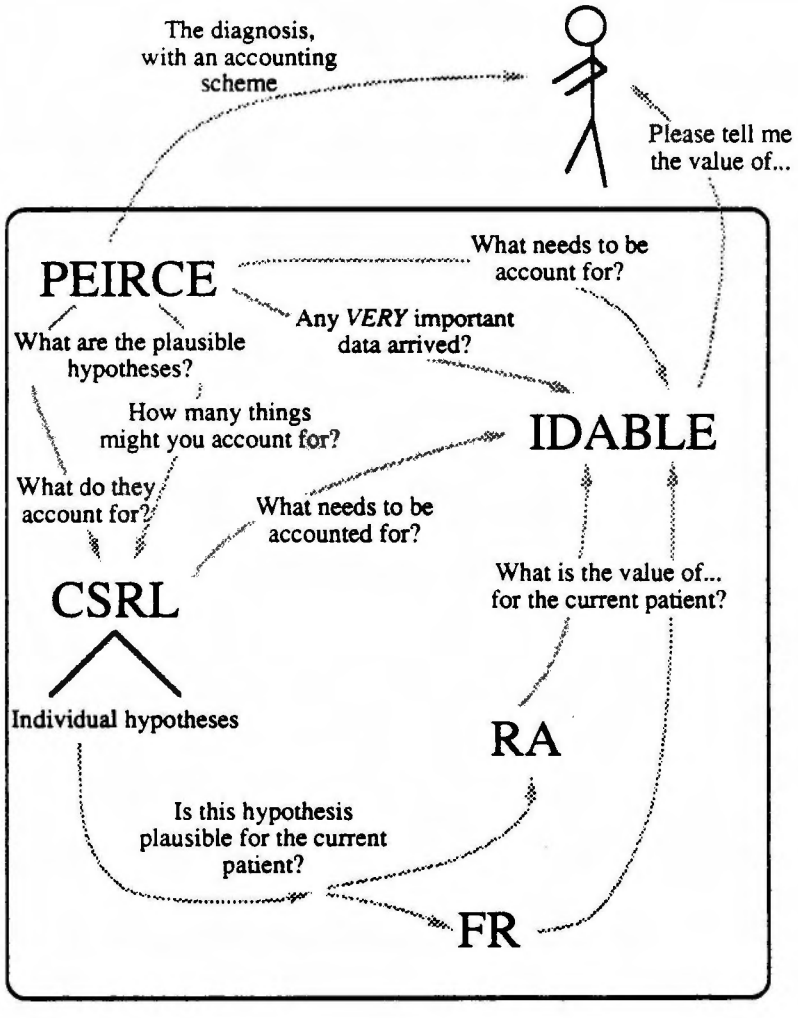


Figure 3: **Communication between generic tasks in MDX2**

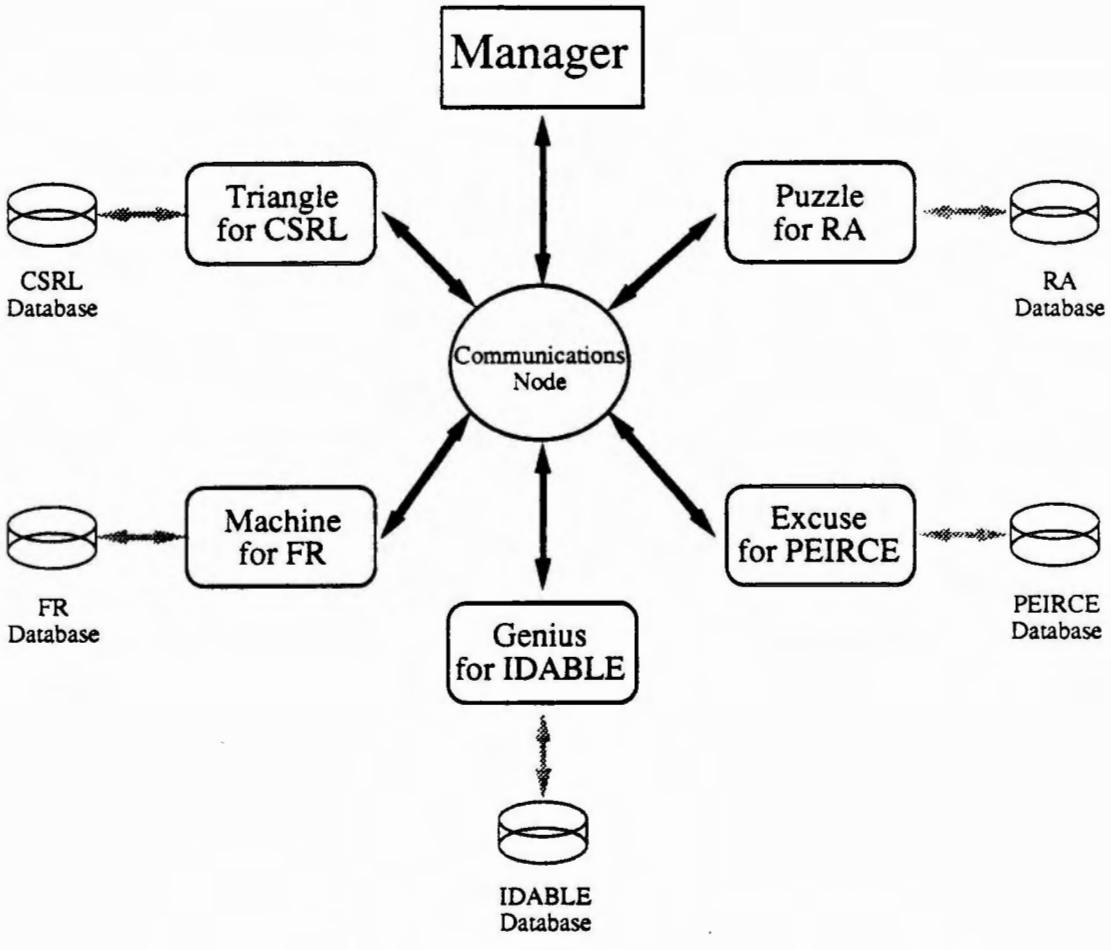


Figure 4: **The KA communications architecture**

9.1 The architecture

Figure 4 shows the main features of the architecture (based upon work in [10]). It has been designed so that each knowledge acquisition module has access to the other modules and the manager. This free access is intended to allow the modules to be used in any order, depending on the structure of the generic tasks. It is the manager which controls this process.

Each knowledge acquisition module works to construct a database of its own type of knowledge. This database can be altered in much the same way as any other database (deletions, updates, amendments).

The modules are linked to each other and to the manager via a communications link. This allows the manager to send messages to any knowledge acquisition module, and any module to send information back to the manager or to any other module.

The manager has overall control of the system. Its job is to invoke knowledge acquisition modules as required. It is able to do this by using a basic outline of the generic tasks and how they communicate. This has to be given to the manager before the program can be run.

This does not mean that the manager controls all knowledge acquisition modules that are called. For example, in Figure 3, CSRL communicates with RA, FR and IDABLE. However, a particular CSRL specialist will only communicate with one of these tasks. This has to be decided upon by the knowledge acquisition program at run time. This decision is made from the results of questions asked of the expert.

9.2 Communication

The communication mechanism has two main functions. The first is to allow communication when knowledge acquisition is being performed. The second is to maintain consistency between the individual databases.

The manager and the knowledge acquisition modules use the communications link in two ways. The messages that they send are either directed towards a particular entity (module or manager) or a general request is put on the communications link. Any module can attempt to deal with the general request; if it succeeds it sends confirmation to the sender of the request; if it fails the request is simply put back on the communications link allowing other modules to try and deal with it.

Consistency is an important problem. Information is very much interlinked between databases. If a change is made in one database, then information in another database may have to be altered or discarded. An example would be if we deleted a CSRL specialist; links to other specialists would have to be altered and so would the generic task used to decide if the specialist is to *establish* or not. The links to other specialists are dealt with internally by *Triangle*; the related knowledge acquisition module to the *establishing* dragon is called via the communication link and if it is not used anywhere else is deleted.

9.3 Example: Acquiring knowledge for MDX2

To test the knowledge acquisition program, it is being used to perform the knowledge acquisition for MDX2. If a system similar to MDX2 can be constructed using the program then it indicates the approach has some promise.

Only three modules have been constructed to date therefore the knowledge acquisition for MDX2 cannot be performed in its entirety. This section gives an example of how the system at present performs the knowledge acquisition for part of MDX2.

Looking at Figure 3 the part of MDX2 that the program can at present account for is the sub-system in the lower part of the diagram: CSRL, RA and FR. The following shows the flow of information:

1. The manager is given access to a file containing the basic structure of the generic tasks.

2. The manager invokes *Triangle* because CSRL is the top level task (in this sub-system); this was known because of the outline of the generic task structure that the manager has. It is a direct call to *Triangle*.

3. *Triangle* requests the list of leaf specialists from the user. Each of these specialists needs a means of deciding if it can *establish* or not. For each specialist a general call is put onto the communications link asking if any knowledge acquisition module can find a way of *establishing* the specialist.

4. One of two things happen next. There are only two knowledge acquisition modules (at present) which are capable of being used to *establish* a specialist.

5. *Puzzle* tries first. It asks the expert if it is possible to decide if the specialists is true by looking at values of different objects. If so, *Puzzle* tries to construct a Recognition Agent from information given by to it by the expert. The name of the Recognition Agent is passed back to *Triangle* so that it can be included as a call from the CSRL dragon.

6. If *Puzzle* is rejected then it puts the general call back onto the communication link. *Machine* picks up this call and asks the expert if a simulation of a model is required to show that the specialist is true. If so, *Machine* produces a model with the help of the expert. The name of the model is passed back to *Triangle* so that it can be included as a call from the CSRL dragon. If not, the call is put back onto the communications link.

   If CSRL receives the call to *establish* because it has gone round the communications link unprocessed, then a stub is left as an empty dragon call which can be processed at a later date.

7. CSRL continues its job (as described in section 4) and for each new specialist it acquires, the above calls are made to the system in an attempt to find ways of *establishing* the specialists.

8. Once CSRL has finished, it sends a message back to the manager saying as much. This allows the manager to either call upon the services of another module or stop the acquisition process.

10 How does this system relate to other systems?

There are a number of different approaches to the automation of knowledge acquisition. This section will first describe the three main categories into which these approaches fall and will then describe where our program fits into this classification. The groups are as follows:

Knowledge acquisition for simple representations. Many of today's knowledge acquisition tools are really only useful in the elicitation of knowledge for performing simple classification tasks. If the problem to be solved is not a classification problem, then this type of knowledge acquisition is not very useful.

Some examples of this type of knowledge acquisition tool are automated repertory grids and pictorial card sort tools.

Knowledge acquisition for application areas. A second strand of knowledge acquisition research has concentrated upon developing tools to elicit knowledge for problem solving systems in particular domains. The idea is that domain-specific tools are necessary for the development of knowledge-based systems with specialised problem solving characteristics.

One example of such a problem-specific knowledge acquisition tool is MORE [11]. MORE is specifically designed to help with the acquisition of knowledge for diagnostic tasks.

This type of tool can only be justified for frequently occurring applications or in high-value situations, because each new type of problem or new domain involves building or modifying tools and hence demand significant implementation effort.

Knowledge acquisition methodologies. These methodologies cover the whole of the knowledge acquisition process, and are applicable to knowledge acquisition in a wide range of domains. KADS [1] is a good example of such a methodology backed up with appropriate knowledge acquisition tools.

While the KADS methodology is not limited to one application area (unlike a system such as MORE), its generality means that the level of automation possible in the KADS knowledge elicitation tools is much lower than was the case with MORE.

At present, given that the structure of the generic tasks is fixed (ready for use as a particular application), the program fits into the second description above — it is a knowledge acquisition system for an application area, acquiring detailed knowledge for use in a particular domain.

The eventual aim of this work is that the architecture will become part of a methodology for knowledge acquisition. This would involve performing preliminary knowledge acquisition to decide upon the configuration of generic tasks; the results of this could be passed directly to the program.

The interpretation tasks of KADS are a good example of how this could be achieved. If the preliminary knowledge acquisition was used to construct an outline of the domain in terms of an interpretation model, this model could be used to identify which tasks are required to perform detailed knowledge acquisition.

11 Acknowledgements

Work by D.R. Pugh in this area is funded by a Science and Engineering Research Council research studentship, number 89307105.

References

[1] Schreiber G, Breuker J, Bredeweg B, Wielinga B. Modelling in KBS Development. *Avignon '87*, pages 283–296, 1987.

[2] Minsky M. A Framework for Representing Knowledge. *The Psychology of Computer Vision*, pages 211–277. McGraw-Hill, 1975.

[3] Bylander T, Chandrasekaran B. Generic Tasks for Knowledge-based Reasoning: The "Right" Level of Abstraction for Knowledge Acquisition. *International Journal of Man-Machine Studies*, 26:231–243, 1987.

[4] Chandrasekaran B. Generic Tasks in Knowledge-based Reasoning: High-level Building Blocks for Expert System Design. *IEEE Expert*, 1:23–30, Fall 1986.

[5] Chandrasekaran B. Generic Tasks as Building Blocks for Knowledge-based Systems: The Diagnosis and Routine Design Example. *The Knowledge Engineering Review*, pages 183–210. Cambridge University Press, September 1988.

[6] Sticklen J. MDX2, An Integrated Medical Diagnostic System. *PhD Thesis*, The Ohio State University, 1987.

[7] Pugh DR, Price CJ. Automating Knowledge Acquisition for Generic Tasks. *Engineering Applications of Artificial Intelligence*, Vol. 3, pages 171–179. Pergamon Press, September 1990.

[8] Punch WF. A Diagnosis System Using a Task Integrated Problem Solver Architecture (TIPS), Including Causal Reasoning. *Phd Thesis*, The Ohio State University, 1989.

[9] Chandrasekaran B. Decomposition of Domain Knowledge into Knowledge Sources: The MDX Approach. *Proc. 4th Nat. Conf. Canadian Society for Computational Studies of Intelligence*, Saskatchewan, May 1982.

[10] Hunt JE, Price CJ. Performing Augmented Model-based Diagnosis. *Presented at the International Symposium on Mathematical and Intelligent Models in System Simulation*, Brussels, September 1990.

[11] Kahn G, Newlan S, McDermott J. MORE: An Intelligent Knowledge Acquisition Tool. *Proceedings IJCAI-85*, pages 581–584, 1985.

GENERIC TECHNIQUES IN EMA: A MODEL-BASED APPROACH FOR KNOWLEDGE ACQUISITION

Susan Spirgi, Dieter Wenger

Swiss Bank Corporation, Hochstrasse 16, 4053 Basel, Switzerland

André R. Probst

IBM Switzerland and University of Lausanne, General Guisan Quai 26, 8048 Zürich, Switzerland

Abstract

The model-based problem solving is a natural extension within the development methodology EMA, because EMA is based on the basic principles: abstraction, decoupling and parallelism. A classification of problem types is made by classifying the solution information of the problems. Model-based problem solving is done by G-TECs (Generic TEChniques). A G-TEC consists of a knowledge model and its realisation and is herewith formal and operational. The knowledge model of a G-TEC and that of an application are based on the same meta-model (modeling language). The use of G-TECs leads to the integration of better problem solving methods in the applications, to a better knowledge acquisition and last but not least to a strong reduction of coding work. Prototype-, meta-model-, result- and model-based knowledge acquisition are brought together to a whole.

1. INTRODUCTION

EMA is a methodology for the development of knowledge-based applications. In this paper the model-based approach for knowledge acquisition is outlined.

Chapter 2 gives a short introduction of the basic concepts and principles of EMA. Chapter 3 introduces the underlying conception of this model-based approach and leads to a classification of problem types. Readers familiar with KADS or other methodologies may read chapter 5 before chapter 4. In chapter 5 EMA is related to other methodologies. In chapter 4 the model-based approach by means of G-TECs is explained in detail. Chapter 6 emphasizes the importance of the developer's ability to work on different levels of abstraction. Chapter 7 concludes this paper.

2. INTRODUCTION OF EMA

EMA [7] is an integrated, active and comprehensive methodology for the development of knowledge based applications. It is integrated because the methodology itself is realized as a knowledge-based application. Due to that it is an active partner for the developer. EMA is comprehensive because information of the whole development cycle are kept and managed and EMA supports the developer in each step of the development cycle even in generation of code (now ART-Code, soon TIRS-Code).

The two most important concepts are the information-set (info-set) or information-type (info-type) and the agent. The concept 'info-set' is based on the concepts 'object', 'object-instance' and 'object-class'. 'Object-instance' and 'object-class' are both objects. An info-set is a set of information-units (info-units). In figure 1 OC-x is a object-class and OI-x1, OI-x2, OI-x3, etc. are object-instances. An info-unit is defined by an object and one attribute. In figure 1 OI-x1@Att-y, OI-x2@Att-y and OI-x3@Att-y are info-units. These info-units can be seen as elements of the info-set OC-x@Att-y.

Sometimes instead of the expression info-set the expression information-type (info-type) is used, because OI-x@Att-y is the type of the info-units OI-x1@Att-y, etc..

An info-set, described by an object and one of its attributes, comprises as elements all info-units with the same attribute as the info-set and with objects which are in an is-a- or in an instance-of-relation to the object of the info-set. Info-sets are info-units. Info-sets comprise info-units and are representatives of a set of info-units.

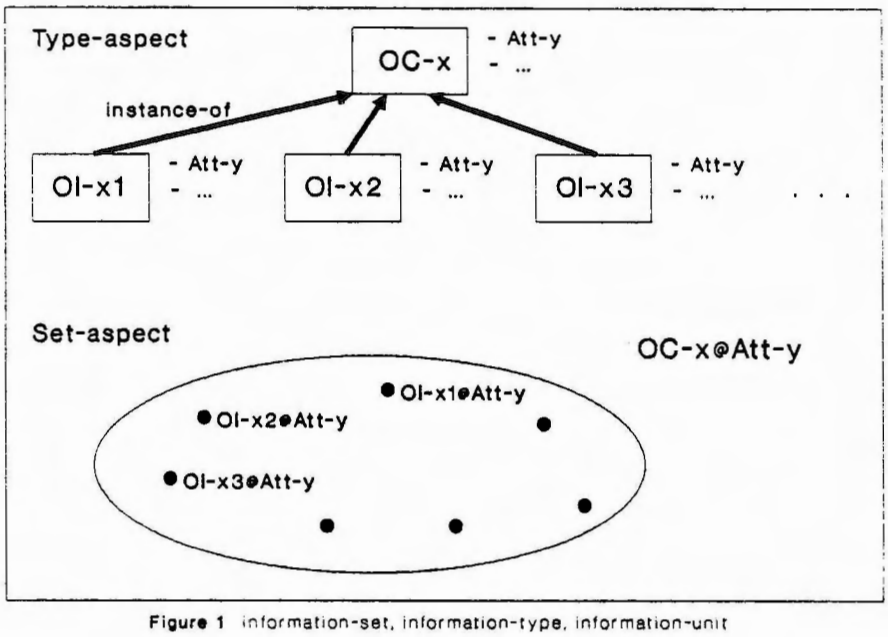


Figure 1 information-set, information-type, information-unit

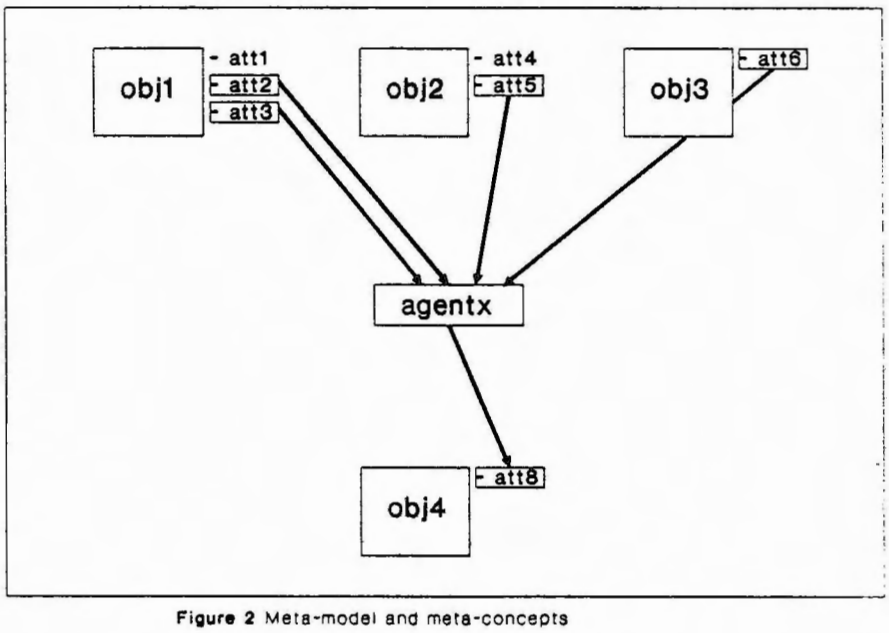


Figure 2 Meta-model and meta-concepts

The concept agent manipulates the info-unit(s) of exactly one info-type (info-set). For this the agent needs some different kinds of information (knowledge), which is expressed by info-sets.

- **activation knowledge:**
  The information specifying when the agent has to become active.
- **input knowledge:**
  The information about input info-sets which have to be processed by the agent for producing the output info-set.
- **constraint knowledge:**
  The information about info-units of the input info-sets which have to be selected.
- **process context knowledge:**
  The information that drives the processing (processing parameters).
- **functional knowledge:**
  The knowledge that is in functional form, specifying how the processing is done. It makes the agent operational.

Due to the fact that an agent manipulates the info-units of exactly one info-type, an agent is related to the object-class of this info-type. The important consequence is that the agent is given the abstraction level of this object-class. Because processing is done by the agents the processing is related to abstraction.

Figure 2 shows an agent that manipulates the info-set obj4@att8. For this it needs info-units of the info-sets obj1@att2, obj1@att3, obj2@att5 and obj3@att6. These info-units belong to the above mentioned kinds of knowledge.

Figure 3 shows how an application is seen. A system is represented in figure 3 where the agents become active depending on some of the info-units and process input info-units to output info-units. The produced output is again taken as input by other agents (black-board principle). In [7] the bottom up support of EMA is described. In the following chapters the top-down support is outlined.

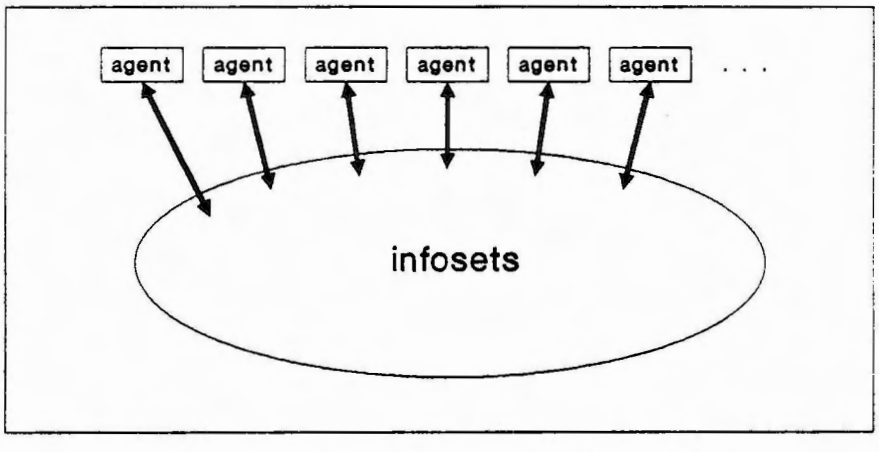


Figure 3 Agents manipulating the info-sets (blackboard principle)

3. PROBLEM SOLVING (PS) IN EMA

The inferencing of information from known information is seen in EMA as PS. Therefore PS may be sometimes trivial sometimes extremely difficult and complex. Figure 4 shows the basic concepts of PS: the problem-description, the PS-procedure and the solution-information (solution-info) of the problem (result of the PS procedure). These basic concepts are related to each other. The problem and the solution-info are related in a 1:1-relationship, whereas the PS-procedure is related in a n:m-relationship to the problem and herewith the solution-info. This means that it is possible that the same solution-info can be built by more than one PS-procedure and that it is even possible that different kinds of problems can be solved by the same PS-procedure.

For making a classification for PS we have to classify at least one of the basic concepts. If we classify the problems, we classify the solution-info and vice versa, because they are related by one to one. It doesn't seem very promising to classify the PS-procedures, because the number is huge and the procedures are hard to describe in a formal way. A formal description facilitates the classification very much.

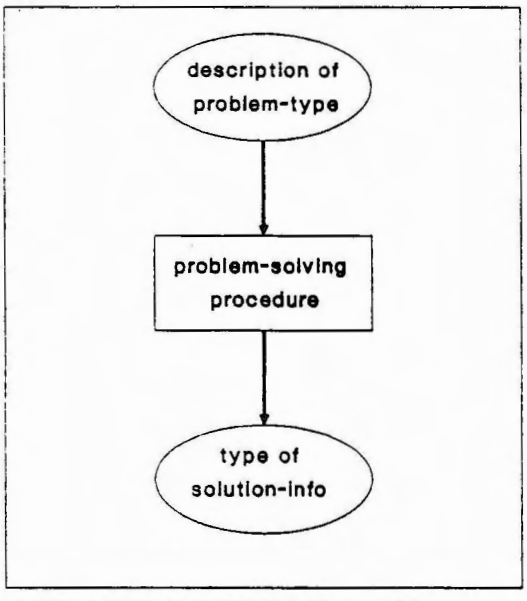


Figure 4 Basic concepts of problem-solving

There are some reasons for classifying the solution-info:

- The solution-info can be entirely expressed with help of info-sets, because it is an information. This means that the things we want to classify can be easily expressed in a formal way. This is an important precondition for classification, because how should it be possible to classify something if it cannot be well described. The solution-info is classified because it is easier to express the model of the solution-info than the model of the problem-description
- For the classification of the solution-info we have a strong criterion: What is the solution-info used for and interpreted as by the user? For this the underlying assumption is that information becomes its meaning and herewith its value as soon as and not before it is used. A strong criterion is another important precondition for classification.

Although the starting conditions for classification are excellent we get into trouble when we analyse and intend to classify the infered informations of different applications. The number of types of information is growing tremendously. Two examples shall be discussed.

There is an information which is interpreted as a ratio of a balance sheet analysis and there is an information which is needed by a user for evaluation (evaluation info). Expressing these two types of information with info-sets gives us:

- balance sheet ratio @ value
- ? @ evaluation

The first info-set, built by the object-class 'balance sheet ratio' and its attribute 'value' (remember an info-set is built by an object-class and one of its attributes), is interpreted as 'the value of a balance sheet ratio' which is considered the class of information. This class of information is limited to a specific domain which is the domain of balance sheet analysis, because the object-class 'balance sheet ratio' is an object-class of this domain. Therefore it can be said that the solution-info 'ratio of balance sheet analysis' is domain-specific.

The second info-set shows a solution-info which is not domain-specific. An infinite number of different objects-classes may have an evaluation. It is not possible to relate these objects to a certain domain. Therefore the solution-info 'evaluation of ...' is a general solution-info.

As a conclusion we get solution-infos (and herewith problem-descriptions) which are general or which are domain-specific. For the classification of solution-infos the concept 'domain' has to be taken into account. So one classification exists, which is domain-independent (general) and one classification for each of the different domains. In other words the huge amount of different kinds of solution-infos can be partitioned into general classes of solution-infos and into the different domain-specific classes of solution-infos.

Figure 5 shows that the domains build an abstraction structure (domain, sub-domain, etc.). Each of these domains have their own classification of solution-infos. This would lead to domain-oriented model-based knowledge acquisition.

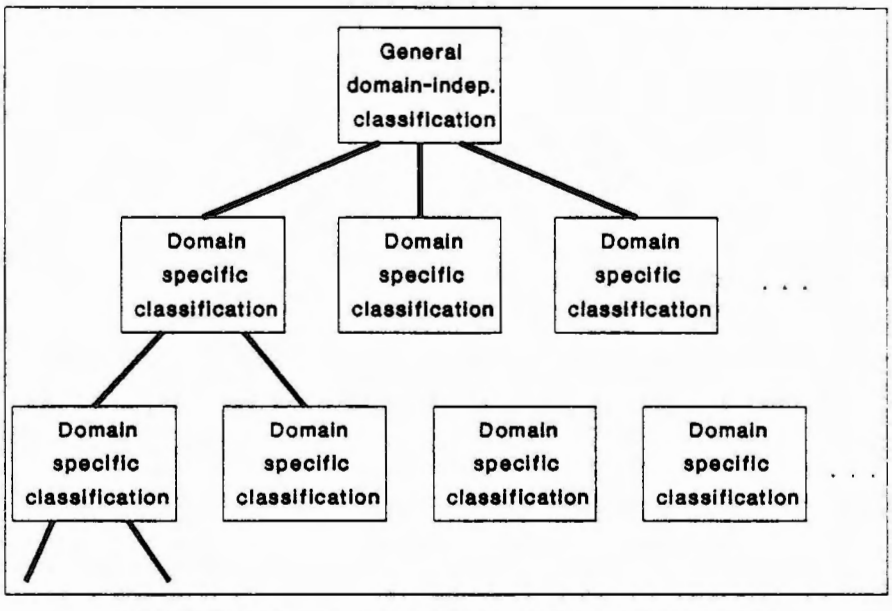


Figure 5 General and domain-specific classification of solution-info

In this paper we restrict ourselves to the classification of general kinds of solution-infos. The current classification which is not yet complete looks like:

- classification info
- prediction info
- evaluation info
- causal info (diagnosis info)
- planning info
- design info
- configuration info
- hint info
- modification-info (advisory info)

Although sometimes prediction info is built by classification a difference is made between 'classification info' and 'prediction info' , because our classification of solution-info is made according to its use.

4. GENERIC TECHNIQUES (G-TECs)

In EMA the building block which should solve a certain kind of problem is called G-TEC. Two kinds of G-TECs exist, the conceptual G-TEC (CG-TEC) and the technical one (TG-TEC). The produced result of a CG-TEC can be classified according to the classification given in chapter 3. A CG-TEC can be related to a certain kind of problem (problem-specific). A TG-TEC is problem neutral (e.g. an optimization technique, statistical technique). TG-TECs are often parts of CG-TECs.

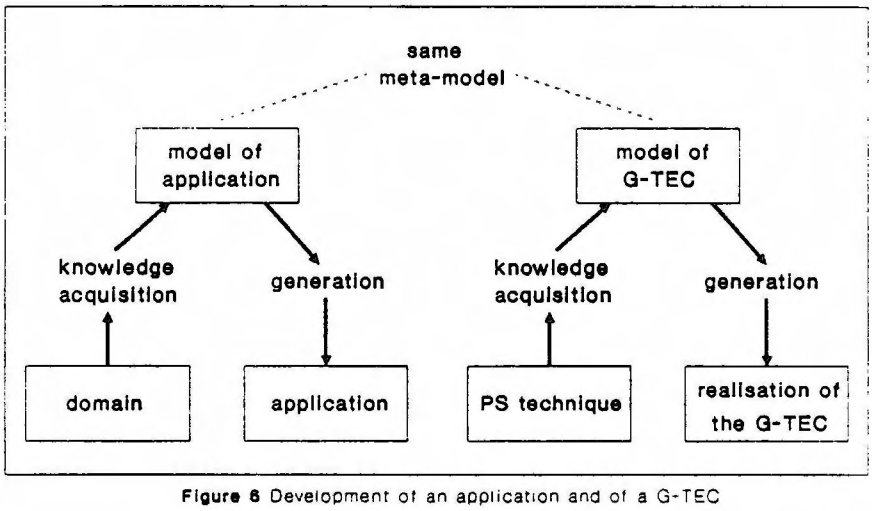


Figure 6 Development of an application and of a G-TEC

4.1. BUILDING A GENERIC TECHNIQUE

The generic technique approach is model-based and has some parallels to generic task [3] [4] and the interpretation model approach in KADS [1] [2].

The developing of G-TECs is principally similar to the developing of applications. At the left hand side of figure 6 the development of an application is shown. The model is made with information which is acquired from the domain. The model is the model of the application and not of the domain. The application is generated with the information of the model (knowledge model).

At the right hand side of figure 6 the development of a G-TEC is shown. The information for the G-TEC model is acquired from a PS-technique which is transformed into a G-TEC. Then the G-TEC is generated with the information of the G-TEC model and some manually provided code which makes the G-TEC operational. A G-TEC consists of its model and its realization (operational part). The models of the G-TECs and the applications are based on the same meta-model (modeling language).

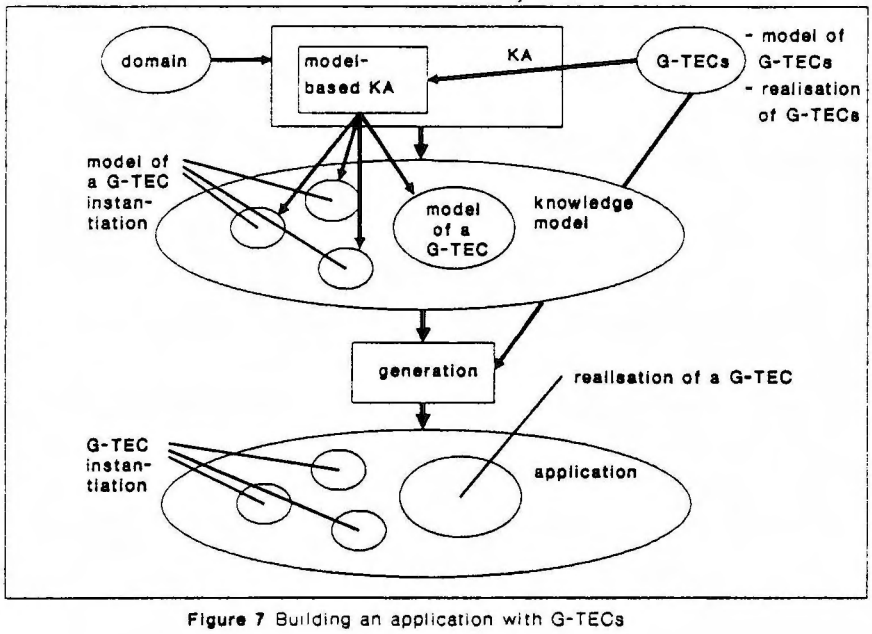


Figure 7 Building an application with G-TECs

4.2. BUILDING AN APPLICATION WITH G-TECs

In EMA besides of model-based knowledge acquisition there are other kinds of knowledge acquisition. From a G-TEC we have its knowledge-model and its realization (operational part). Figure 7 shows how an application is developed with G-TECs. Integrating a G-TEC in an application means to make one ore more instantiation(s) of a G-TEC. The models of the G-TECs and their instantiations become a part of the knowledge model. When the application is generated, the instantiations of the G-TECs are created and integrated in the application. The G-TEC part of an application is obtained without any programming (apart from application-specific methods), because the entire processing of a G-TEC instantiation is inherited from the G-TEC (see also figure 9 and 10).

In connection with the sisyphus project of EKAW91 the development of a whole application is described in [8].

4.3. EXAMPLE OF A CG-TEC: CONDEV

CONDEV is a technique for the condensation of evaluations. A frequently occuring problem is the evaluation of a certain object (a person, company, etc.). It is assumed that information about different partial aspects of this object exists and that this information can be related to evaluations (partial evaluations). The problem is to condense these partial evaluations to intermediate evaluations and to the final evaluation. The mechanism should be robust against incompleteness of information about this object.

The knowledge model of a G-TEC is built with the meta-concepts of EMA. The model of the G-TEC CONDEV can be described with two object-classes, some attributes and two agents (figure 8).

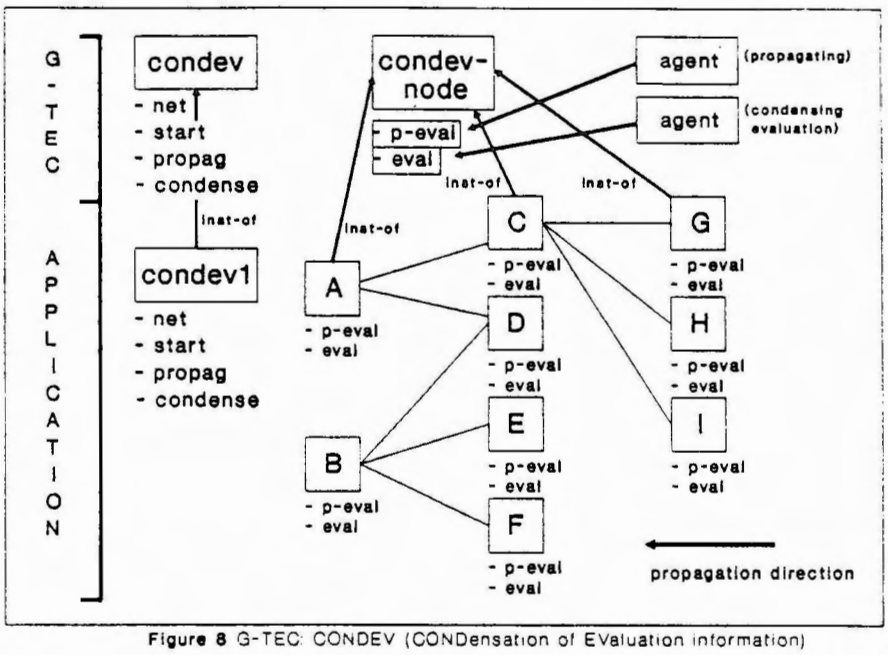


Figure 8 G-TEC: CONDEV (CONDensation of EValuation information)

Figure 8 shows the CG-TEC CONDEV within an application. The upper part of figure 8 is the model of CONDEV and is independent of the application. The lower part is the model of an instantiation of CONDEV and is specific to the application.

The principle of the condensation of evaluation is the following:

Each node of the net (A, B, C, D, E, F, G, H and I are instances of condev-node) has the attributes 'p-eval' and 'eval' which are inherited from the object-class 'condev-node'. The attribute 'p-eval' contains primary and propagated partial evaluations which consist of an average and of a certainty. Primary partial evaluations are the evaluations which have not been created by propagation. They are the input information of CONDEV. There is an agent which propagates the partial evaluations from one node to the node on the next higher level (from the right to the left in figure 7). The second agent of the G-TEC CONDEV builds the evaluation (attribute 'eval') which is based on the partial evaluations of the same node. This evaluation remains local and only the partial evaluations are propagated. The processing is specific to CONDEV which means that the processing is done independent of the net-structure which is specific to the application.

In the CG-TEC CONDEV the attribute 'net' of the object-instances of 'condev' contains the information about the net-structure.

The attributes 'propag' and 'condense' are method-attributes which means their values are method-names (function-names). The method 'propag' is used by the propagating, the method 'condense' by the condensing agent. Some standard methods are provided that differ in the weighting mechanism. The developer selects one of them. In case he needs a different mechanism he can customize a standard method.

4.4. OTHER G-TECs

The number of G-TECs is not limited. All techniques that do any processing may be transformed into G-TECs; e.g. machine learning , reasoning , neural net, operation research techniques, etc.. Very often some techniques can be taken together to a class of G-TECs and classes of G-TECs to more general classes and we get a generalisation structure for the G-TECs. This generalisation can be extremely useful, because some attributes and agents can be attached to object-classes on a higher

level of abstraction; e.g. there are classes of neural net techniques with agents that are specific to the object-class 'neural-net'.

4.5. SETTING UP THE RELATIONSHIP BETWEEN G-TECs AND PROBLEM CLASSIFICATION

Although the G-TECs are modeled and constructed according to their nature (machine learning techniques belong together) we have a different criterion for applying them. We classify according to the kind of produced result. We classify whether the produced information is an evaluation, a planning, a classification information, etc.. The criterion is the following question: What is the produced result used for and interpreted as by the user?

4.6. MODEL-BASED KNOWLEDGE ACQUISITION WITH HELP OF G-TECs

1. A required result is identified in the task-domain and expressed with info-types.
2. The type of the required result is defined with help of the PS classification. The question which should be asked and which represents the criterion for selection is: What is the produced result used for and interpreted as by the user?
3. If at least one CG-TEC exists, knowledge acquisition driven by the underlying CG-TEC and supported by EMA is carried out. The decision whether a CG-TEC is selected depends on the availability of the information, which must be acquired in the task domain, and herewith on the success of the knowledge acquisition process. A successfull knowledge acquisition process leads to a model of the CG-TEC instantiation which may be integrated into the knowledge model of the application (see also figure 10).
4. The selected CG-TEC implies different info-sets. Some of these info-sets have to be produced first. The info-sets become required results. For getting a required result go to 1..

All these steps are carried out cooperatively by developer and EMA.

EMA is able to produce hints to draw the developer's attention to inconsistent situations as e.g. missing input or context information and EMA helps with visualisation features, so e.g. the managing of the net-structure in CONDEV.

It should not be forgotten that very often no CG-TEC exists for a required result. Then the part of the knowledge model for obtaining this required result has to be built from scratch with the help of other kinds of knowledge acquisition provided by EMA [7].

5. POSITIONING OF G-TECs

In this chapter EMA and its G-TEC approach is positioned to other approaches. A comparison is first made to KADS, because KADS is currently a well-known approach. Then a positioning is made following the structure of 'Models, methods, roles and tasks: Many labels - one idea!'[5].

5.1. COMPARISON EMA - KADS

KADS has a **multi-layer conceptual model** and a **design model**. EMA only has a **knowledge model** which comprises the two KADS models. EMA has **views** which correspond roughly to the layers of KADS. Views are special representations of the knowledge model and show specific aspects: so the **information view** (IV) shows the information aspect, the **local agent view** (LAV) the processing aspect and the **global agent view** (GAV) the control aspect. The **domain layer** corresponds to the IV (not vice versa), the **inference layer** to the LAV and the task and **strategic layer** to the GAV. The **knowledge source** of KADS corresponds to an **agent** and the **meta-class** would be in EMA expressed by an **object-class**, which has a 'has-role' relation to other object-classes. The **problem solving method** of KADS is called in EMA a **G-TEC**. The **interpretation model** corresponds to the

knowledge model of a G-TEC. A **task** in KADS corresponds to an **activity** in EMA. EMA has also the concept **task** but it is used in a completely different manner. It is the task that is carried out cooperatively by user and application. In EMA only one modeling level exists. The **implementation level** of KADS is the **prototype level** (application) in EMA.

5.1. POSITIONING OF EMA ACCORDING TO [5]

The term model in EMA contains the information about an application or a G-TEC and is therefore something abstract because information is also abstract. Aspects of the model are made visible with help of different views (representations).

The problem solving method is modeled with the same meta-model (modeling language or formalism) as the application, but the model of the problem solving method is on a higher abstraction level, because the object-classes within the model are on a high level of abstraction and therefore also the attached processing, because it obtains the abstraction level of the object-classes (see chapter 2.). The G-TEC models are problem-specific and herewith independent of the application. Compared to Generic Task and its interaction problem, where knowledge cannot be represented free of assumption of how it will be used, in EMA the information which is used is based on the same meta-model. Information becomes knowledge as soon as and not before it is used for solving a problem. The same information may be used for solving different problems.

EMA is in opposite to the life cycle principle of KADS, where analysing of knowledge and expertise can and must be done before design and implementation, based on a **knowledge model and parallel prototyping approach** which means that the different parts of an application are at one moment on different levels of concreteness. The levels of concreteness span the range between 'no model at all' to 'operational'. The different parts are developed in parallel but decoupled in the sense of their state. This approach stands also in opposite to rapid prototyping. It demands some qualities of the underlying principles of the methodology which are: decoupling of the processing building blocks, abstraction of the information and of the processing and parallelism. This is achieved by a data-driven black-board based agent and object-oriented approach.

Similar to KADS the knowledge model can be developed from scratch (meta-model-, result-, prototype-driven) and with the help of G-TECs.

The corresponding terms to the terminology developed in [5] are:

- Task in [5]: Task has in EMA a completely different meaning. User and application fullfill a task in cooperation (cooperative problem solving). The term activity corresponds to task in [5]. An activity may be carried out by the user or the application and is caused by an event. An activity is a part of the whole task.
- Problem solving method in [5]: The G-TECs are the problem solving methods in EMA. They consist of a model and an operational part (realisation of the G-TEC).
- Action of the problem solving method in [5]: The agent corresponds to this term.
- Modeling level in [5]: The model of all of the G-TECs are based on the same meta-model (modeling language), as the knowledge model of an application. EMA generates the code of the implementation environment (ART, soon also TIRS). The operational part of a G-TEC, which is problem-specific and herewith independent of the application, consists of code of the implementation environment.
- Conceptual model in [5]: In EMA no distinction between conceptual and design model is made. It is called the knowledge model and contains the whole information-based knowledge about an application. The functional knowledge which makes the application operational is not contained in the knowledge model but attached to it.
- Control knowledge in [5]: Control knowledge is one kind of knowledge used by the agent for doing its job at the right moment within an activity (activity in the sense of EMA).

In [5] three hypothesis have been set up:

Hypothesis 1:

It is useful to describe problem solving behaviour on a more adequate, abstract level than is offered by general purpose knowledge representation languages.

In EMA the problem solving behaviour is described by the model of the G-TEC which is based on the same meta-model (modeling language) as the knowledge model of the application. Moreover a coded part is attached to the model which makes the problem solving operational.

Hypothesis 2:

Models of problem solving behaviour can be specified in a problem specific, but application-task independent way.

Figure 9 gives an overview over model-based problem solving in EMA.

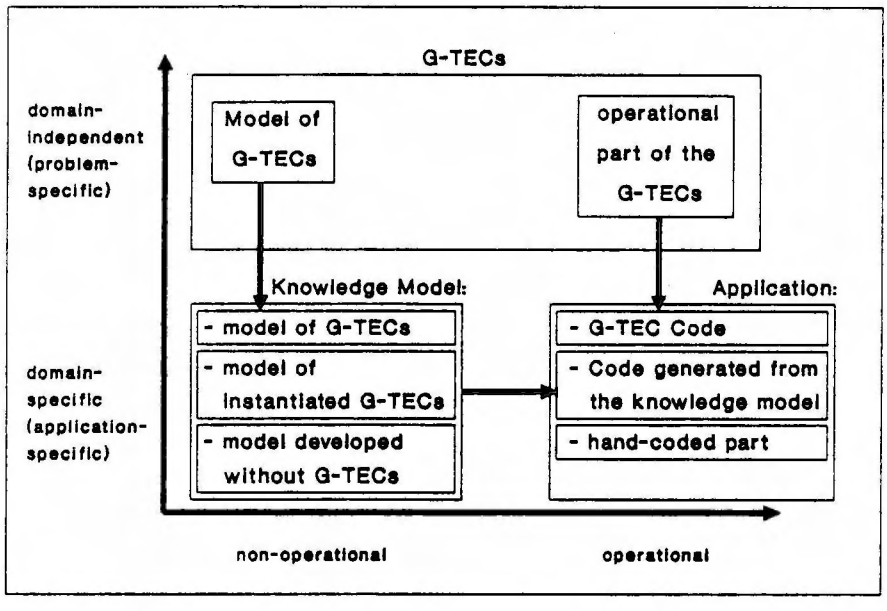


Figure 9 Relationship between G-TEC and Application ordered according to specialisation and operationality

EMA provides a modeling language (meta-model) for the modeling of the G-TECs. Moreover the G-TECs are operational. The different G-TECs can be easily combined, because they are parts of the same knowledge model and based on the same meta-model (modeling language).

For choosing the right problem solving method (CG-TEC) EMA gives a support with the help of a problem taxonomy and of the G-TEC-specific KA facilities: EMA proposes one ore more G-TEC(s) and shows the user the needed information for applying a CG-TEC. The user checks whether the needed information is available in his task domain. If so the CG-TEC may be applied.

Figure 9 shows the model-based problem solving in EMA.

Hypothesis 3:

Models guide the knowledge acquisition process and structure the knowledge base.

The support for knowledge acquisition provided by EMA is described with help of figure 10.

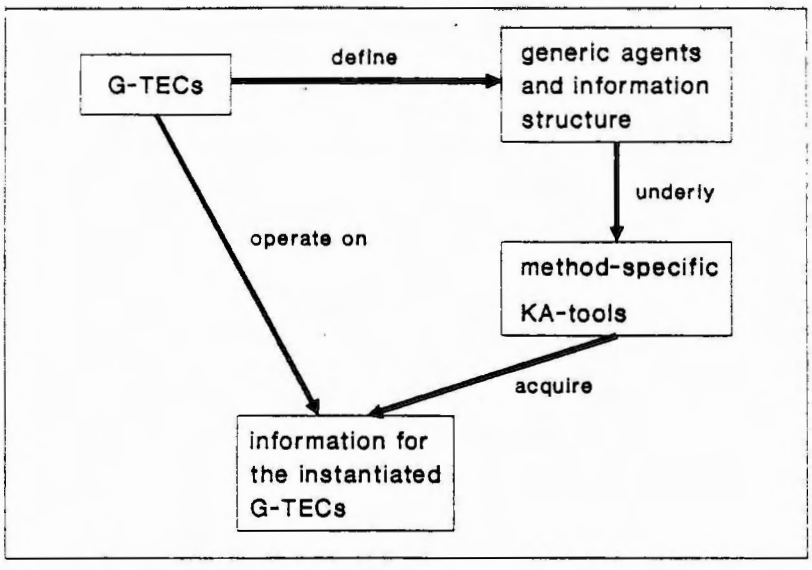


Figure 10 Relationship between G-TEC and Knowledge Acquisition

The knowledge base is structured by the knowledge model, because the information-based knowledge of the knowledge-base is generated from the knowlege model and because the operational knowledge is attached to the information-based knowledge of the knowledge model.

6. LEVELS OF ABSTRACTION FOR THE DEVELOPMENT

With EMA the developer is working on different levels of abstraction.

The most abstract level is the G-TEC-level. The next level is the agent-info-set-level, then the level of the knowledge-based tool (ART or TIRS) and at last the implementation level of the tool (language C).

7. CONCLUDING COMMENTS

The G-TEC approach in EMA is a concrete realisation of a part of the vision that is formulated in [6] about generic models and in [5] about model-based knowledge acquisition.

The G-TEC approach is a natural extension of EMA due to the basic principles of EMA which are decoupling of processing, parallelism and abstraction.

EMA has become an integrated, comprehensive and active methodology which supports the developer from the first to the last step in application development. Nevertheless there is still a lot of work to do: more techniques must be transformed into G-TECs; EMA must be optimized; it must become more efficient and more stable; it must become more developer friendly, etc..

REFERENCES

[1] Breuker J. (ed), Wielinga B., Schreiber G., de Greef P., de Hoog R., van Someren M., Wielemaker J., Billault J.P., Davoodi M., Hayward S., (1987) Model Driven Knowledge Acquisition: Interpretation Models, Del A1, Esprit p1098, University of Amsterdam

[2] Breuker J., Wielinga B., (1989) Models of Expertise in Knowledge Acquisition, Memorandum 103 of the VF-project "Acquisition of Expertise"

[3] Bylander T., Chandrasekaran B., Generic tasks for knowledge-based reasoning: the "right" level of abstraction for knowledge acquisition, Int. J. Man-Machine Studies (1987) 26, pp. 231-243

[4] Chandrasekaran B., Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design, IEEE Expert (Fall 1986), pp 23-30

[5] Karbach W., Linster M., Voss A., (1990) "Models, methods, roles and tasks: many labels - one idea?", Knowledge Acquisition 2, Academic Press Limited, pp. 279-299

[6] Shadbolt N., Wielinga B., (1990) Knowledge based knowledge acquisition: the next generation of support tools, EKAW-90, Current trends in knowledge acquisition, IOS

[7] Spirgi S., Probst A., Wenger D.,(1990) "Knowledge Acquisition in a methodology for knowledge-based applications", proceedings of JKAW '90, pp. 382-397

[8] Spirgi S., Wenger D., Probst A., Sisyphus Project EKAW-91

Interaction types and chemistry of generic task models

Jan Treur

Vrije Universiteit Amsterdam
Department of Mathematics and Computer Science
Artificial Intelligence Section
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
email: treur@cs.vu.nl

Introduction

In the design of a modular knowledge-based system a description of the interactions between the modules is an important issue. To give a detailed specification of these interactions, both data-modelling aspects (information) and process-modelling aspects (control) have to be taken into account: *what* data have to be transferred from one module to another module and *when and how* these interactions should be carried out. In this paper we will concentrate on the data-modelling aspects.

We analyse what types of interactions can be distinguished between modules in a modular knowledge-based system. Such an analysis needs a well-defined notion of a modular knowledge-based system. Our framework for DEsign and Specification of Interacting REasoning modules (DESIRE) provides such a notion (some references: [4], [7], [8], [9]). Therefore our analysis will be carried out in relation to this framework.

The knowledge-based modules in a modular system are called *reasoning modules*. The modules that are specified by a conventional formalism (such as an algorithm description, or a data base description) are called *conventional modules*. As far as interactions between modules are concerned, two important notions should be explained: the notion of semantic identification and the notion of reflective interaction. Both notions play an essential role in our framework DESIRE. We will explain them below.

Each module is related to (refers to) a specific fragment of the world. We assume it uses a language to describe the current state of this fragment of the world (also called world situation). This language will be assumed declarative; in fact we use syntactical expressions of (sometimes many-sorted) predicate logic. However, for convenience we will restrict the semantics of this language to propositional semantics. The state of the fragment of the world that is concerned can be described by truth values of the closed atomic propositions of the module's language (for shortness these are called atoms). So, at any moment in a session of the system the information state of a module is a partial truth assignment to the atoms of the module, and this information state is viewed as a specific partial description of the current state of the fragment of the world that is concerned. By reasoning or some other type of processing in the module this partial description can be refined to a more complete description of the fragment of the world that is concerned.

A state in a modular knowledge-based system is viewed as a collection of partial descriptions of fragments of the state of the world (world situation). Here some of these fragments of the world may overlap and that is the point where interactions may enter the scene. If an atom **a** in module A and an atom **b** in module B both refer to the same item in the world (see Fig 1), then in the system one may want to specify this by means of a *semantical identification* of **a** and **b**. In our approach such a semantical identification is established by specifying *transformations* that link atoms of different modules to each other, and that describe the semantical connection between the atoms that are linked in terms of the relation between the truth values of them. For example, if **a** is the atom denoting that a patient has fever and **b** denotes that the patient is without fever, then **a** and **b** can be identified with each other, with a semantical relation given by the truth table **<<true, false>, <false, true>>** .

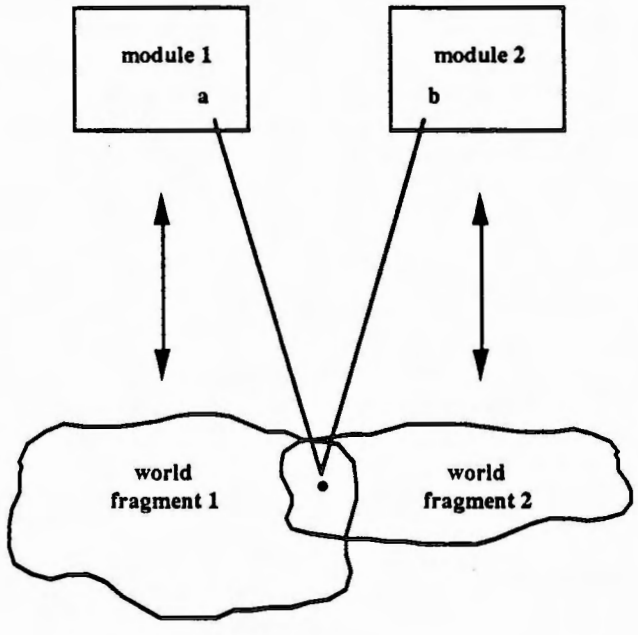


Fig 1 Semantical identification

I.e. if **a** is true then **b** is false and if **a** is false then **b** is true (and conversely). For convenience, we will not discuss here the role of truth values **unknown** in this story.

Of course, in most systems one will use the same names for atoms with essentially the same meaning; in that case the identification as sketched is just syntactical equality. However, if modules are written by other developers, or re-used from other systems, in different modules one may have different names for the same items in the real world. In that case nontrivial semantical identifications are needed to enable each module to make use of the information that is obtained by other modules.

A second important notion in DESIRE is the notion of reflective interaction. In many applications involving more complex knowledge-based systems a reasoning pattern is modelled that essentially is reflective (see [7], [9]). This means that during the reasoning sometimes the subject of the reasoning is not a fragment of the external world but a fragment of the state of the

system (i.e. of the reasoning process) itself. In DESIRE reflective patterns are modelled in a quite integrated manner. If a module reasons about some fragment of the system then this is treated similar to reasoning about a fragment of a (changing) world situation. We refer the reader to [4] and [7] for a more detailed introduction to reflective interactions and the role they play in complex reasoning patterns. Since DESIRE supports reflective interactions and these interactions essentially affect the dynamic behaviour of a system, combinations of these types of reflective interactions are related to different types of dynamic reasoning patterns performed by a system.

The interaction types as distinguished will be illustrated by small example systems to explain in what manner some given combination of interaction types induces a dynamic reasoning pattern. Examples of the following generic reasoning patterns are modelled:

- reasoning with dynamic hypotheses in diagnosis
- explicit reasoning about observations (tests) to be carried out in diagnosis
- (non-monotonic) reasoning with dynamic assumptions in a simple scheduling task
- reasoning with causal knowledge to reject dynamically assumed hypotheses

It turns out that from the relatively low number of interaction types as distinguished here, by combining them a large variety of generic reasoning patterns or generic task models can be configured. In a metaphoric sense this phenomenon can be considered as a chemistry of generic task models. It will be shown that actually the interaction types are the most important building blocks for a generic task model, and not the modules. If we extend the metaphor somewhat further, the interaction types play a role that may be compared to the role of the types of bindings in chemistry.

In section 2 we define our classification for interaction types. It is based on distinctions between object-level and meta-level inputs and outputs (as summarized in section 1) and distinctions within meta-level inputs and outputs. In section 3 a number of examples explain the use of these interaction types as building blocks to model the reasoning tasks as mentioned above. These examples do not include systems that integrate reasoning and learning. In section 4 it is discussed what kind of interaction types can be used to cover this type of system as well.

1 The roles of object level information and meta-level information

Knowledge-based modules have a very distinctive structure: a knowledge base component and an inference engine component. Starting point for our classification of interaction types is the distinction originating from the fact that according to its specific structure each module stores two types of information, i.e. the object-level information state and the meta-level information state (see [4]). Since these are generic characteristics we consider it reasonable to make the distinction between incoming object-level information and meta-level information not in the module itself but in the specification of the transformation by which interaction takes place.

In the description of a transformation we take into account both roles that are played by the transferred information: the role of this information in the domain module of the transformation and the role in the co-domain module of the transformation. So in our approach the information about the two roles that are played by the transferred information are part of the transformation specification. This enables us in the first place to obtain a very transparent and declarative generic view on the notion of a module. In the second place we can use the interaction types to

describe a system in a generic manner in more detail (generic task models; see [4]). Moreover, it enables us to make a classification of types of behaviours of such generic architectures based on a classification of interaction types (see section 2).

This results in the following four global types of interactions (or data-transformations) between modules:

object-interaction	from object-level of module A to object-level of module B
meta-interaction	from meta-level of module A to meta-level of module B
upward reflection	from meta-level of module A to object-level of module B
downward reflection	from object-level of module A to meta-level of module B

In Fig 2 these global types of interactions are depicted. We sketch some examples of them:

1. *From object information state to object information state (object-transformation)*
A very simple example of this type of interaction is provided by a reasoning process that consists of two phases. For instance this may occur in diagnosis, especially in heuristic classification, if some initial data on symptoms are known. In the first phase data-abstraction is performed, say by module A, classifying the observables (symptoms) to more abstract statements. In the second phase, by module B these abstract data are used to perform a heuristic match to a certain proposal for (abstract) hypotheses to investigate. In fact in such a type of interaction one may model the reasoning by one module as well: the reasoning performed in the first phase, just continues in the second phase.

2. *From object information state to meta-information state (downward transformation)*
An example of this is a system, in which module A draws conclusions that are statements about the inference strategy of module B. In strategic reasoning patterns such an interaction may occur. For instance module A may derive **possible_target(h)**, i.e. that **h** is a suitable goal for module B. The interaction is carried out by reading the conclusions of module A and changing the control parameters of module B accordingly; in this case the goal list is adapted to include **h**. In the meta-level information state of module B this can be expressed by the truth value **true** of the meta-fact **target(h, determine)**. After such a transformation this target will affect the inference procedure of module B. We will treat a more detailed example in section 3.2. Also other meta-facts may be involved. For instance, module A may derive that the assumption that the object-level atom **a** is false could be made (by module B), i.e. it draws the conclusion that **possible_assumption(a, neg)** is true. As a result of a downward transformation, in the meta-information state of B the meta-fact **assumption(a, neg)** is given the truth value **true**.

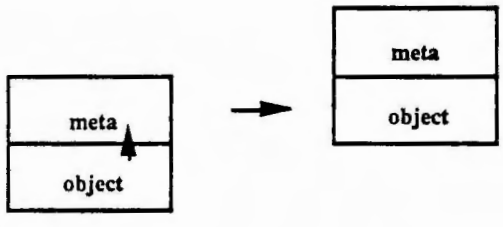
3. *From meta-information state to object information state (upward transformation)*
In some sense this is the converse of the type described in 2. above. In this case module B needs data about the meta-information state of module A. For instance in the meta-information state of a module A the truth value **false** for the meta-fact **known(a)** may be represented: the atomic



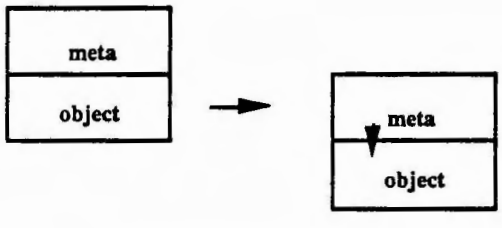
object transformation



meta-transformation



upward transformation



downward transformation

Fig 2 Global types of interactions

statement a is currently not known in module A. This is meta-information on the object level information state of module A. There may exist a module B that uses this fact to derive some heuristic consequences of this. In section 3.2 a more detailed example of this type of interaction will be treated.

4. *From meta-information state to meta-information state (meta-transformation)*
An example of this is obtained by using a module A that is performing goal-directed reasoning. If the meta-information state of module A contains the truth value **true** for the fact **to_query(a)** that currently the atomic statement **a** is a query for any other module (or the user), then an interaction of this type occurs if this truth value is transformed to the truth value **true** of the meta-fact **target(a)** in the meta-information state of module B that asserts that **a** is a goal of module B.

All interactions take place by means of transformations of information states. These are used to translate data in the format that is required in the next module. A transformation may be defined by a mapping from the set of information states it is acting on (domain), to the set of the information states that result from the transformation (co-domain). Often one would like to keep facts that already were obtained in a module in earlier activations during a session. Therefore we also view the earlier information state as an argument of the transformation: the second argument represents this information state, while the first argument is the output information state from a previous module. Based on the information as represented in the latter, the former is changed by the transformation.

For instance, assume module A is related to the set of possible information states P_1 and module B is related to P_2. Then an interaction from A to B is described by a transformation α as follows:

$$\alpha(N, M_1) = M_2$$
$$\alpha : P_1 \times P_2 \rightarrow P_2$$

Here N represents the information state (the part of it that may be used as output) of module A after activation and M_1 represents the information state of module B resulting from the earlier activations of module B. The outcome M_2 of this transformation takes over the role of information state in module B as played earlier by M_1. By this definition of a transformation it is possible to integrate certain forms of revision (in the sense of belief revision or truth maintenance) as well. In that case N contains the information leading to revision of the information state M_1 to the updated information state M_2. We may say the information state N *induces* via α a transformation of M_1. The way in which revisions may propagate to other modules is an issue that will be described in more detail elsewhere.

We describe information states by partial models; our definition of transformation may be compared to the role of reflection principles as used in [12]. By an example, in [7] the relation between both approaches is shown.

2 Classification of interactions

From the viewpoint of the dynamic behaviour of a system, the global classification as given in section 2 is a rather rough classification. In particular, there are different types of interactions falling in the class of downward reflections which result in completely different types of behaviour, such as setting a target, or making an assumption. In fact, the information as expressed by our generic meta-predicates at the meta-level of a module refers to various (generic) aspects of reasoning, concerning for example:

- the targets of a module (meta-predicate **target**)
- the assumptions made in a module (meta-predicate **assumption**)
- the epistemic state of the module (meta-predicates **known, true, false**)
- the requests of a module (meta-predicate **to_query**)

Each of these types of information plays a different role in the reasoning process as a whole, especially if it is involved in interactions between modules.

At the moment it is hard to establish which aspects should be added to this list to make it complete in some sense, i.e. such that it covers all relevant generic aspects of reasoning. However, the experience we have until now shows that even by combining interaction types based on the (maybe restricted) set of generic meta-predicates, described in the list above, a surprising number of interesting generic types of dynamic behaviour can be realized. It turns out that in any case we can use this list to create nontrivial reflective reasoning patterns such as:

- reasoning with dynamic hypotheses, as needed in diagnostic tasks (section 3.2)
- reasoning about which observations to carry out (section 3.3)
- reasoning with dynamic assumptions (including default reasoning), as needed, for instance, in design tasks (section 3.4)
- reasoning about goal transformation and goal refinement, as needed in design tasks (for instance, see [9])
- hypothesis rejection based on reasoning with causal knowledge (section 3.5)
- reasoning about action in dynamic process control (see [11])
- reasoning from different viewpoints (see [11])
- hypothetical reasoning (see [11])
- reasoning about uncertainty and incompleteness (see [11])

Therefore, in this paper we base our classification on the generic meta-aspects as listed. We investigate how the basic interaction types related to these meta-aspects can be used to built generic reasoning patterns that arise by combining them. For this process of combination, as a metaphor the term chemistry (of generic task models) may be used, which explains the title of the paper. The meta-predicates as mentioned can be divided in a natural manner in input meta-predicates and output meta-predicates related to a module; these correspond respectively to the co-domain and domain of a transformation:

input meta-predicates:	**target; assumption**	(co-domain of transformation)
output meta-predicates:	**to_query; known, true, false**	(domain of transformation)

If these are assigned to respectively the horizontal axis and the vertical axis, then we obtain the matrix of combinations as depicted in Fig 3. Here the different types of interactions in this matrix are explained as follows:

- *object interaction* (type O-O)
  This is the very simple type of transformation that carries over (object) conclusions from module A to (object) input facts of a second module B. As a consequence, these can be used as facts in the reasoning of module B.

co-domain of transformation

domain of transformation	object level	target set	assumption set
object level	object interaction	target setting	assumption making
epistemic information	epistemic upward transformation	epistemic-target meta-transformation	epistemic-assumption meta-transformation
request information	request explicitation	request-target meta-transformation	request-assumption meta-transformation

Fig 3 Matrix of interaction types

- *target setting* (type O-T)
  This type of downward transformation transforms (object) output facts of module A into meta-facts of module B the form **target(a, t)** where **a** is an atom name of module B and **t** is a target type. As a consequence, in the subsequent activation of module B these targets control the reasoning process (backward reasoning).

- *assumption making* (type O-A)
  In this type of downward transformation (object) output facts of module A are transformed into meta-facts of module B the form **assumption(a)** where **a** is a literal name of module B. As a consequence, in the reasoning of module B such **a** can be used as if they were facts, although they are labeled at the meta-level that retraction of them may take place.

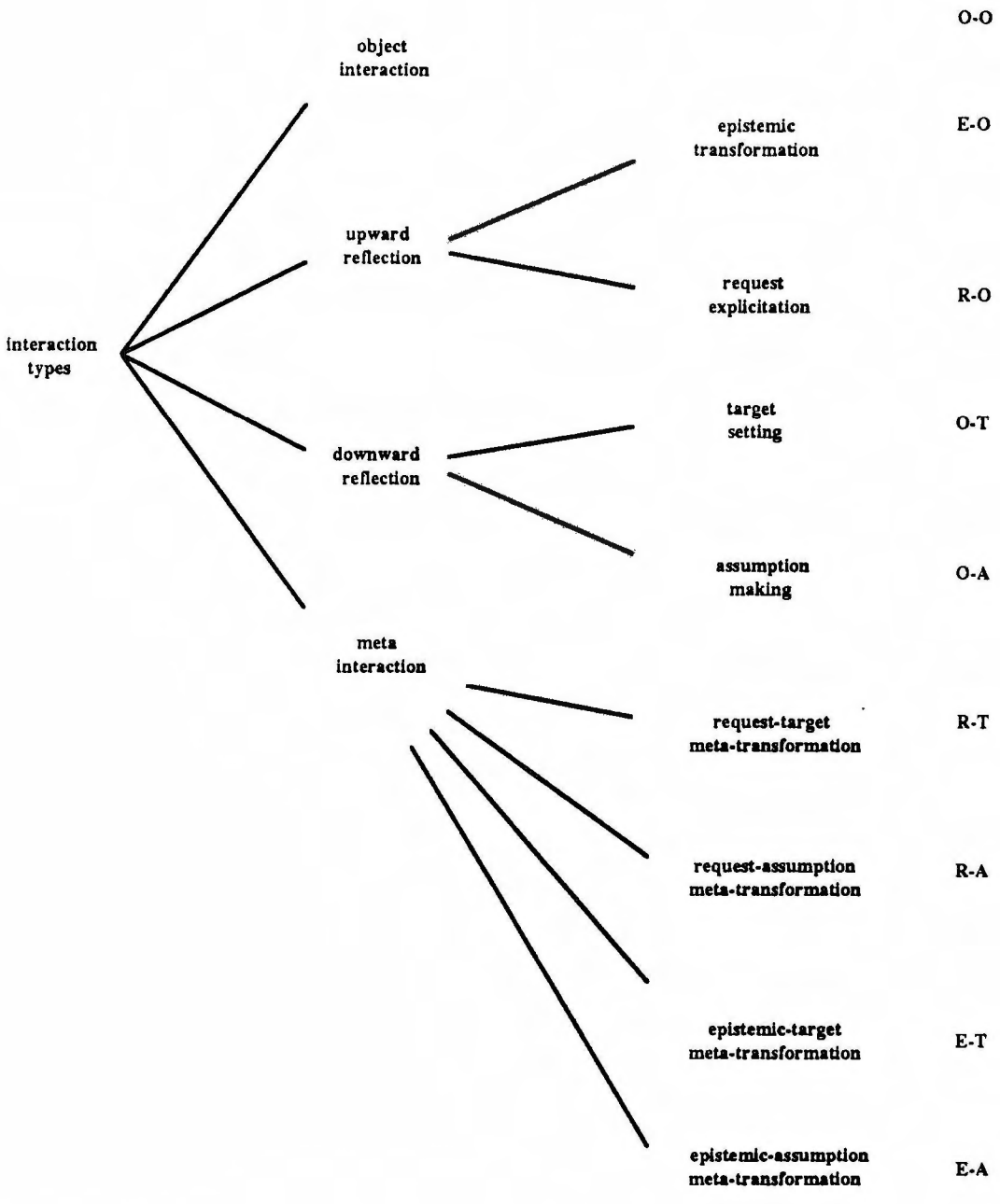


Fig 4 Taxonomy of interaction types

- *epistemic upward transformation* (type E-O)
  This is a type of upward reflection by which meta-facts of module A of the form known(a), true(a), false(a), where a is an atom name of module A, are transformed into object input facts of module B. Subsequently these facts can be used in the reasoning of module B.

- *request explicitation* (type R-O)
  Also this is a type of upward reflection. In this case meta-facts of module A of the form to_query(a, t), where a is an atom name of module A and t is a target type, are

transformed into object input facts of module B. Again, subsequently these facts can be used in the reasoning of module B. For instance this reasoning could obtain a conclusion on how this request may be handled by the system.

- *request-target meta-transformation, or request passing* (type R-T)
  This is the most relevant meta-transformation we have distinguished until now. By this transformation meta-facts of module A of the form **to_query(a, t)**, where **a** is an atom name of module A and **t** is a target type, are transformed into meta-facts of module B of the form **target(a, t)**. As a consequence, these targets control the reasoning during the subsequent activation of module B.

- *request-assumption meta-transformation* (type R-A)
  For example, this type of meta-transformation can be used if one wants to transform the meta-facts on requests of module A into assumptions of module B in order to try to show that this leads to contradictions. So one can model request-answering by the method of indirect proof using this type of interaction.

- *epistemic-target meta-transformation* (type E-T)
  For instance this type of meta-transformation can be used if one wants to transform the epistemic meta-fact **¬known(a)**, expressing that the atom **a** is unknown in module A, to a target meta-fact **target(a, determine)** of module B in order to try to find knowledge on **a**. Another example is when the meta-facts **known(a)** are transformed to targets of the form **target(a, determine)** of module B, in order to verify whether module B obtains the same truth value for **a** . This pattern may play a role in hypothetical reasoning. It is described in more detail below.

- *epistemic-assumption meta-transformation* (type E-A)
  An example of the use of this type of interaction is when everything derived as true by module A is interpreted by module B as an just assumption.

In a different form of presentation this provides the taxonomy of interaction types as depicted in Fig 4. After having defined the interaction types we consider, in section 3 we will present some simple examples of system architectures where these interaction types play a role.

3 Examples

The examples as treated in this section show how the interaction types are used as building blocks to build generic task models performing some required complex reasoning task. For reasons of presentation, we will not give the system architectures in the form of generic task models but in the form of (a semi-formal description of) example systems, so that the reader can have an impression of a concrete reasoning pattern. We assume that, after having understood these reasoning patterns it will not be difficult for the reader to abstract from the example systems the corresponding control structures and generic task models.

3.1 Object interaction

This interaction type in the first place models different phases in a reasoning process; an example is the following. Here first symptom abstraction takes place and then from these abstract data hypotheses are derived. For convenience we only give short descriptions and no complete specifications.

module 1 : *symptom abstraction*
This is a reasoning module. Its signature consists of:
- set of input atoms (observables) $S_0 = \{s_1, s_2, s_3, s_4\}$
- set of output atoms (hypotheses) $H_0 = \{i_1, i_2\}$

The knowledge base is given by:

$$s_1 \rightarrow i_1$$
$$s_2 \rightarrow i_1$$
$$s_3 \rightarrow i_2$$
$$s_4 \rightarrow i_2$$

The inferences are done data-driven.

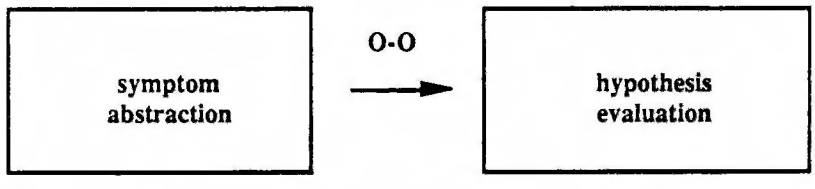


Fig 5 Simple type of O-O interaction

module 2 : *hypothesis evaluation*
This is a reasoning module. Its signature consists of:
- set of input atoms (observables) $S_0 = \{i_1, i_2\}$
- set of output atoms (hypotheses) $H_0 = \{h_1, h_2\}$

The knowledge base is given by:

$$i_1 \wedge i_2 \rightarrow h_1$$
$$i_1 \wedge \neg i_2 \rightarrow h_2$$

The inferences are done data-driven.
Here the inputs needed for hypothesis evaluation module are provided as outputs by the symptom abstraction module in a trivial manner.

3.2 Dynamic hypothesis selection

The modular system consisting of the two reasoning modules and the world situation module as described below performs a complex pattern of reasoning with dynamic hypotheses. It is based on the description of hierarchical classification and heuristic classification in the literature (for instance see [2]) and the logical analysis we made of these diagnostic reasoning methods (see [7], [10]).

module 0: *external world*
This is modelled as a conventional module. The signature of the module consists of:
- set of input atoms (observables): $S_0 = \emptyset$
- set of output atoms (hypotheses): $H_0 = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8\}$

The information state of this module is given by all truth values of facts the user puts in in the system.

module 1: *hypothesis evaluation*
This is a reasoning module; its signature consists of:
- set of input atoms: $S_1 = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8\}$
- set of output atoms: $H_1 = S_1 \cup \{h_1, h_2, h_3, h_4, h_5\}$

The knowledge base KB_1 consists of:

$$s_2 \wedge s_4 \rightarrow h_1$$
$$\neg s_3 \wedge s_5 \rightarrow h_2$$
$$s_1 \wedge \neg s_6 \rightarrow h_3$$
$$\neg s_4 \wedge s_7 \rightarrow h_4$$
$$\neg s_5 \wedge s_8 \rightarrow h_5$$

All input atoms are meant as requestable. The interaction with the external world (module 0) can answer these queries. The inferences will be done goal-directed (backward chaining), where the target set will be determined dynamically by the module for hypothesis selection.

module 2: *hypothesis selection*
The signature of this module consists of:
- set of input atoms $S_2 = \{ \text{true}(s) \mid s \in S_1 \} \cup \{ \text{false}(s) \mid s \in S_1 \}$
- set of output atoms $H_2 = \{ \text{posshyp}(h) \mid h \in H_1 \}$

The knowledge base KB_2 consists of:

$$\text{true}(s_2) \wedge \neg \text{known}(s_4) \rightarrow \text{posshyp}(h_1)$$
$$\text{false}(s_3) \wedge \neg \text{known}(s_5) \rightarrow \text{posshyp}(h_2)$$
$$\text{true}(s_1) \wedge \neg \text{known}(s_6) \rightarrow \text{posshyp}(h_3)$$
$$\text{false}(s_4) \wedge \neg \text{known}(s_7) \rightarrow \text{posshyp}(h_4)$$
$$\text{false}(s_5) \wedge \neg \text{known}(s_8) \rightarrow \text{posshyp}(h_5)$$

The inferences will be done data-driven (forward chaining); no targets will be set.

Interactions between the modules
The following interaction types (see Fig 6) occur:

1. Epistemic transformation α_1 (type E-O)
This transformation translates epistemic meta-information on observables of S_1 (as acquired by module 1 during its goal-directed reasoning, stored in the object information state of module 1, and expressed using meta-predicates **true, known, false**) to information on these observables as described by object level atoms of S_2, and adds this to the input information state of module 2.

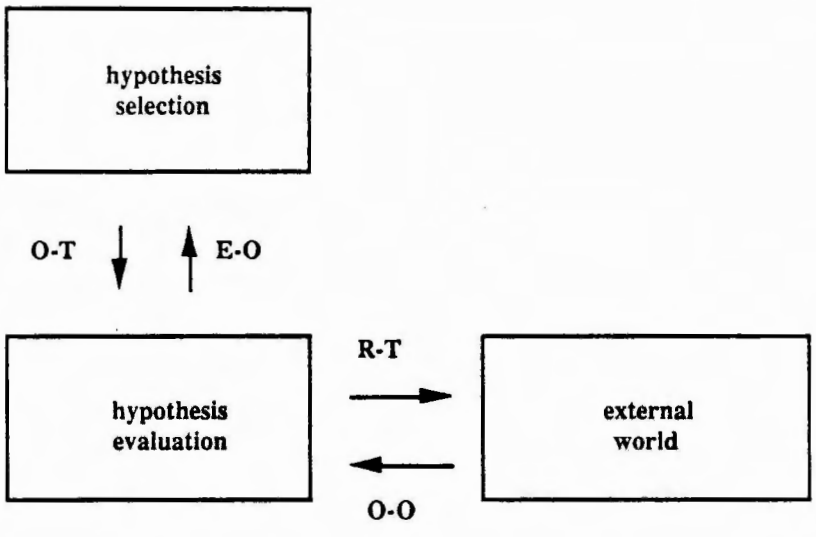


Fig 6 Dynamic hypothesis selection

2. Target setting α_2 (type O-T)
A conclusion of module 2 that a statement of the form **posshyp(h)** is true (as stored in the output object information state of module 2) is transformed by α_2 to the truth value **true** of the meta-fact **target(h, determine)** of the meta-level of module 1.

3. Request passing α_3 (type R-T)
The transformation α_3 translates queries from module 1 into targets of module 0: a truth value **true** of the meta-fact **to_query(a, t)** is transformed into the truth value **true** of the meta-fact **target(a, t)**, where **t** is one of **confirm, reject, determine.**

4. Object interaction α_4 (type O-O)
The transformation α_4 translates results of these queries to input facts in the object input information state of module 1.

The alternation of reasoning with module 1 and reasoning with module 2 (and making observations by module 0) can be managed by some simple supervisor rules. This provides a complex diagnostic reasoning process in which explicit strategic elements occur as meta-level reasoning (i.e., in module 2). In Fig 7 a trace of the reasoning process using these transformations

is shown. Notice that we simplified the meta-predicates **target**, **to_query** and **derived** by omitting the second argument (all target types are meant as **confirm**).

Trace of an example session

Suppose our example system has a session with a situation with the following (observable) input information state, corresponding to $< s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8 >$:

$$< 0, 1, 1, 0, 1, 1, 1, 0 >$$

and we start the system by giving the truth value of s_2 to module 1, as an initial observation.

module 0 *module 1* *module 2*

$p : [s_2], [\,]$
$q : [s_2], [\,]$

t: [true(s_2)]
u: [true(s_2), posshyp(h_1)]

$p_1 : [s_2\,], [\text{target}(h_1)]$
$q_1 : [s_2\,], [\text{target}(h_1), \text{to_query}(s_4]$

$v_1 : [\,], [\text{target}(s_4)]$
$w_1 : [\neg s_4], [\text{target}(s_4), \text{derived}(s_4)]$

$p_2 : [s_2\,, \neg s_4], [\text{target}(h_1)]$
$q_2 : [s_2\,, \neg s_4, \neg h_1], [\text{target}(h_1), \text{derived}(h_1)]$

t_1: [true(s_2), false(s_4)]
u_1: [true(s_2), false(s_4), posshyp(h_4)]

$p_3 : [s_2, \neg s_4, \neg h_1], [\text{target}(h_4)]$
$q_3 : [s_2, \neg s_4, \neg h_1], [\text{target}(h_4), \text{to_query}(s_7)]$

$v_2 : [\neg s_4], [\text{target}(s_7)]$
$w_2 : [\neg s_4, s_7], [\text{target}(s_7), \text{derived}(s_7)]$

$p_4 : [s_2\,, \neg s_4, s_7, \neg h_1], [\text{target}(h_4)]$
$q_4 : [s_2\,, \neg s_4, s_7, \neg h_1, h_4], [\text{target}(h_4), \text{derived}(h_4)]$

Fig 7 Trace of an example session

Then the complex reasoning results in the trace as given in Fig 7. The process stops if one of the hypotheses has been confirmed. Here information states are denoted by the list of atomic facts and negations of atomic facts that are true in them.

Declarative conclusions and procedural actions
The declarative conclusions of module 2 are transformed to putting a hypothesis as a target for module 1. In fact this is a procedural action, with the chosen target as a parameter; module 2 supports the decision about which parameter value for this action may be chosen. Transformations from declarative conclusions to procedural actions are essential for knowledge-based systems for which dynamic behaviour is important. In combination with the appropriate tools, it gives the opportunity to specify some aspects of this dynamic behaviour in a declarative, knowledge-based manner.

3.3 Including a reasoning module for explicit observation selection

The example as described above can easily be extended by a fourth module; this provides a modular system with four modules, where both hypothesis selection and observation selection are performed by explicit reasoning processes (see Fig 8). The other two modules remain the same. This time the inferences in module 1 will be done data-driven, since now targets will be provided for this module. The reasoning pattern as created in this way, essentially is the reasoning pattern of SIX (generic task model for Strategic Interactive eXpert systems; see [3], [10]).

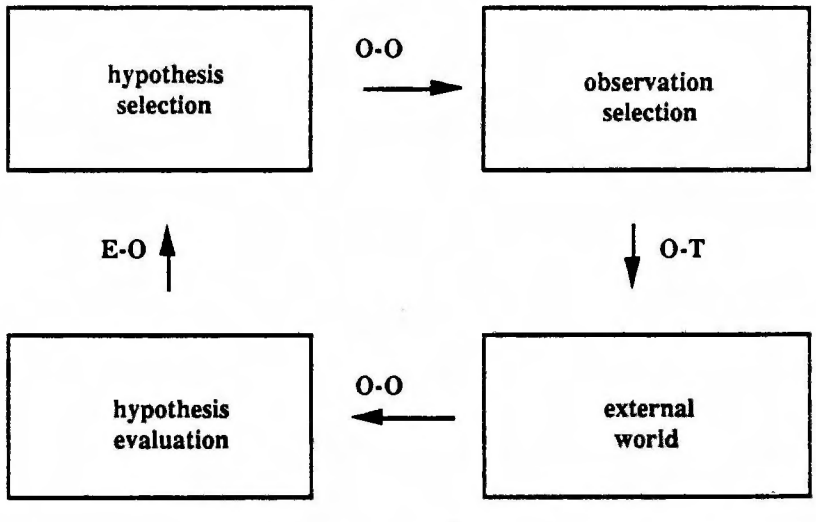


Fig 8 Extension by explicit observation selection

Interactions between the four modules
The following new transformations are added, replacing the old α_2 and α_3.

2. Object interaction α_2 (type O-O)
The interaction is defined in such a way that this time the truth value of a conclusion of module 2 of the form **posshyp(h)** is simply given as the same truth value of the same object input fact to module 3.
3. Target setting α_3 (type O-T)
A truth value **true** of a conclusion of module 3 of the form **possobs(s)** is transferred to the truth value **true** of the meta-fact **target(s, determine)** to module 0.

Carrying out α_3 gives the a truth value true to some meta-fact **target(s, determine)**. As a result, if module 0 is activated the truth value of the observable **s** as indicated is added to the object output information state of module 0. After this activation this truth value is passed to the object input information state of module 1 (by the old transformation α_4). Processing the cycle consisting of respectively module 0, module 1, module 2, module 3 provides a complex diagnostic reasoning process in which this time explicit strategic elements on both hypothesis selection and observation selection occur as meta-level reasoning. The supervisor will have knowledge about this cycle.

Goal-directed reasoning by backward chaining.
Notice that in the simple example given above, in fact the goal-directed inference regime (by backward chaining) has been decomposed into two modules reasoning data-driven (forward). This provides a logical description of goal-directed reasoning as a special ('compiled') case of reflective reasoning. A disadvantage of an approach using backward chaining is that it is not possible to represent explicit knowledge on which observation to do first. The only possibility is to "express" these types of knowledge in the order of rules in the knowledge base and of conditions in rules. In practice this often leads to knowledge bases that are in fact complicated PROLOG-like programs. If maintenance is needed in such a knowledge base, the whole reasoning procedure as it is coded in it will have to be reconsidered. This is not in the spirit of the original (declarative) idea of a knowledge base.

3.4 Dynamic assumption selection

The following bi-modular system exhibits a complex reasoning pattern to generate dynamic assumptions based on heuristic knowledge. It can be considered as a schedule generator for a task in which three subtasks A, B and C are distinguished, with the requirement that A and C *must*, and B and C *may not* occur simultaneously. If we use a schedule in two time periods (morning and afternoon, for instance), then we can give the following logical description of possible schedules:
a) Basic information on elements of the schedule (subtasks)
s_1 : subtask A occurs in time period I
s_2 : subtask B occurs in time period I
s_3 : subtask C occurs in time period I
The negation of an atom represents the fact that the subtask occurs in period II.
b) The requirements about the schedule
h_1 : subtask A and subtask C occur in the same time period
h_2 : subtask B and subtask C do not occur in the same time period

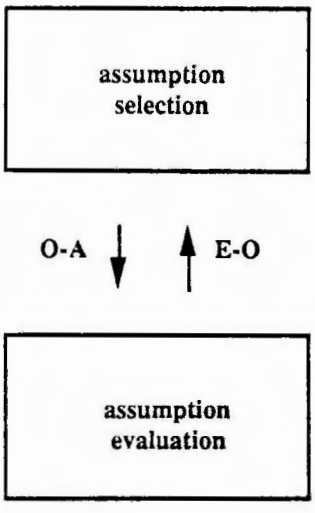


Fig 9 Dynamic assumption selection and evaluation

module 1 : assumption evaluation (ORM)
Signature of the module:
- set of input atoms $S_1 = \{s_1, s_2, s_3\}$
- set of output atoms $H_1 = \{h_1, h_2\}$

Knowledge base KB_1 :

$$s_1 \wedge s_3 \rightarrow h_1$$
$$\neg s_1 \wedge \neg s_3 \rightarrow h_1$$
$$s_2 \wedge \neg s_3 \rightarrow h_2$$
$$\neg s_2 \wedge s_3 \rightarrow h_2$$

No targets will be provided, so the inference type is forward chaining. One or more of the observable atoms s_1, s_2, s_3 are provided by the "outside world" of this module (the user, or the heuristic reasoning module).

module 2: assumption selection (MRM)
Signature of the module:
- set of input atoms $S_2 = \{true(s_1), true(s_2), true(s_3)\} \cup \{false(s_1), false(s_2), false(s_3)\} \cup \{known(s_1), known(s_2), known(s_3)\}$
- set of output atoms $H_2 = \{possass(s_1, pos), possass(s_2, pos), possass(s_3, pos)\} \cup \{possass(s_1, neg), possass(s_2, neg), possass(s_3, neg)\}$

Knowledge base KB_2 :

$$true(s_1) \wedge \neg known(s_3) \rightarrow possass(s_3, pos)$$
$$false(s_1) \wedge \neg known(s_3) \rightarrow possass(s_3, neg)$$
$$true(s_3) \wedge \neg known(s_2) \rightarrow possass(s_2, neg)$$
$$false(s_3) \wedge \neg known(s_2) \rightarrow possass(s_2, pos)$$

No targets will be provided, so the inference type is forward chaining. The truth values of input atoms come, after a transformation, from module 1.

The generic architecture is depicted in Fig 9.

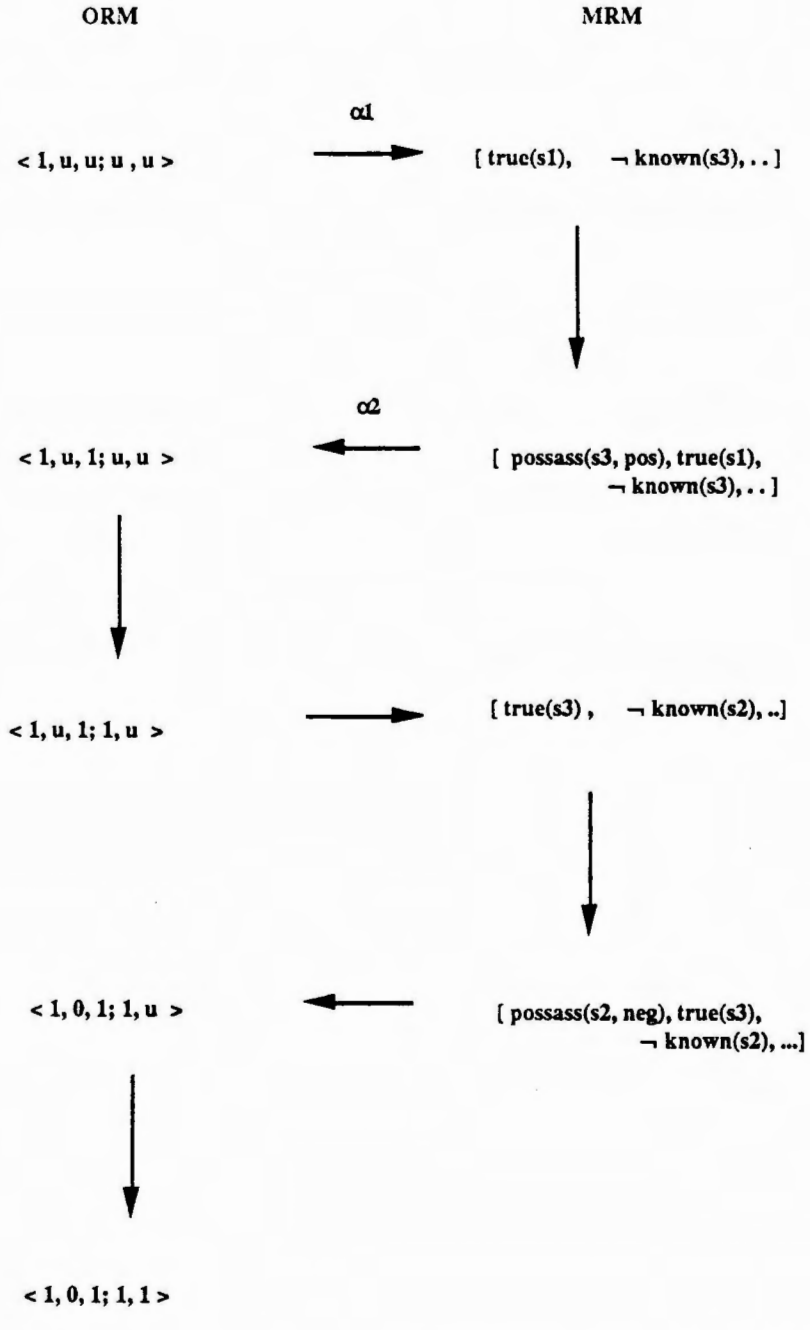


Fig 10 Reasoning trace for scheduling problem

Interactions between the modules.
The modules interact in two directions:

1. Epistemic transformation α_1 (type E-O)
There is an epistemic upward reflection α_1 where the truth values of meta-facts of the form true(s_1), known(s_1), et cetera, are transformed to truth values of object input facts of the meta-module (which are expressed in the same language).

2. Assumption making α_2 (type O-A)
There is an assumption-making downward reflection where truth values of conclusions of the meta-module of the form **possass(a, sign)** are transformed to truth values of meta-facts **assumption(a, sign)** in the object module. Thus assumptions on the schedule are made.

The process starts by providing truth values for a subset of $\{s_1, s_2, s_3\}$, and continues until either the object-module infers all hypotheses $\{h_1, h_2\}$ or the meta-module can provide no more assumptions. Meanwhile, reasoning within the object module is alternated with reasoning within the metamodule.

In theory, there are two possible solutions. The system as described above is capable of generating these solutions when given either "s_1 is true" or "s_1 is false" as a starting assumption. Fig 10 represents the trace starting with the assumption that s_1 is true; i.e. it is assumed that A is scheduled in period I. Using this assumption the system infers that C is in period I and B is in period II. The reasoning pattern can be used to model nonmonotonic reasoning, as is shown in [6].

3.5 Diagnostic reasoning based on causal knowledge

As a combination of the interaction types illustrated in the three sections above, in this section we show how diagnostic reasoning based on causal knowledge can be modelled. In fact the reasoning pattern could be described by "hypothesis rejection". The idea is based on the notion of falsification as studied in the philosophy of science (for instance see [5]). The standard interpretation of the empirical cycle, for instance as depicted in Fig 11, is that the lower arrow is obtained by assuming the chosen hypothesis for a while, and then by causal knowledge deriving what observable facts would follow from it. These derived observable facts are chosen for verification by observation. If the observation contradicts the derived fact, then the assumed hypothesis is rejected. In the other case it is confirmed; however, this gives no certainty since it is not clear whether there may be other observations possible contradicting the hypothesis.

We model this reasoning pattern as follows. Replace the module 1 of section 3.2 (containing anti-causal knowledge) by the following module 1 containing causal knowledge:

module 1: *causal consequences of hypotheses*
This is a reasoning module; its signature consists of:
- set of input atoms: $S_1 = \{h_1, h_2, h_3, h_4, h_5\}$
- set of output atoms: $H_1 = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8\} \cup S_1$

The knowledge base KB_1 consists of:

$h_1 \rightarrow s_2$
$h_1 \rightarrow s_4$
$h_2 \rightarrow \neg s_3$
$h_2 \rightarrow s_5$
$h_3 \rightarrow s_1$
$h_3 \rightarrow \neg s_5$
$h_4 \rightarrow \neg s_4$
$h_4 \rightarrow s_7$
$h_5 \rightarrow \neg s_5$
$h_5 \rightarrow s_8$

No input atoms are requestable. The inferences will be done data-driven (forward chaining).

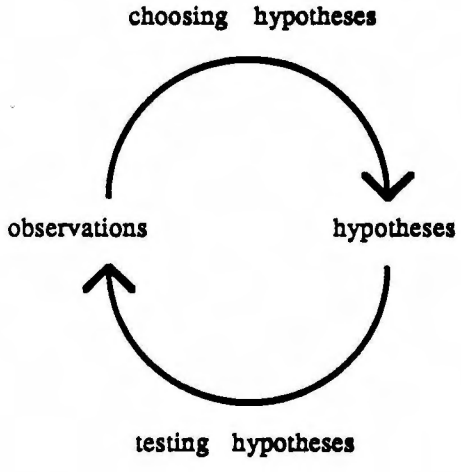


Fig 11 Empirical cycle

The hypothesis selection is done by module 2 of section 3.2., and also module 0 of the same section is used to do observations. With these three modules almost all of the reasoning can be modelled, if suitable interaction types between them are chosen. As an additional module a kind of decision module is needed, where it is decided whether or not to reject a chosen hypothesis. The criterion for this may be chosen very simple and uniform: if, on the basis of an assumed hypothesis **h** module 1 derives **s** while module 0 answers that **s** is not true, then **h** must be rejected; otherwise not. This module is described in predicate logic as follows. Here, for shortness we denote

$S = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8\}$
$H = \{h_1, h_2, h_3, h_4, h_5\}$

The meaning of the predicate **cc_derived** is "derived by the causal consequences module"; the predicate **observed** means "observed in the external world".

module 3: hypothesis rejection

This is a reasoning module; its signature consists of:

- input signature:

$$S_3 = \{\text{focus}(h) \mid h \in H_1\} \cup \{\text{cc_derived}(s) \mid s \in S_1\} \cup \{\text{observed}(s) \mid s \in S_1\}$$

- output signature:

$$H_3 = \{\text{rejected}(h) \mid h \in H_1\}$$

Using the notation from predicate logic, the knowledge base KB_3 consists of the following rule (that is considered as universally quantified):

$$\text{focus}(h) \wedge \text{cc_derived}(s) \wedge \neg \text{observed}(s) \rightarrow \text{rejected}(h)$$

Interaction types

We discuss the types of the transformations as depicted in Fig 14.

1. Object interaction α_1 (type O-O)

Here the truth values of the output atoms **posshyp(h)** of the hypothesis selection are transformed into the same truth values of the input atoms **focus(h)** of the hypothesis rejection module.

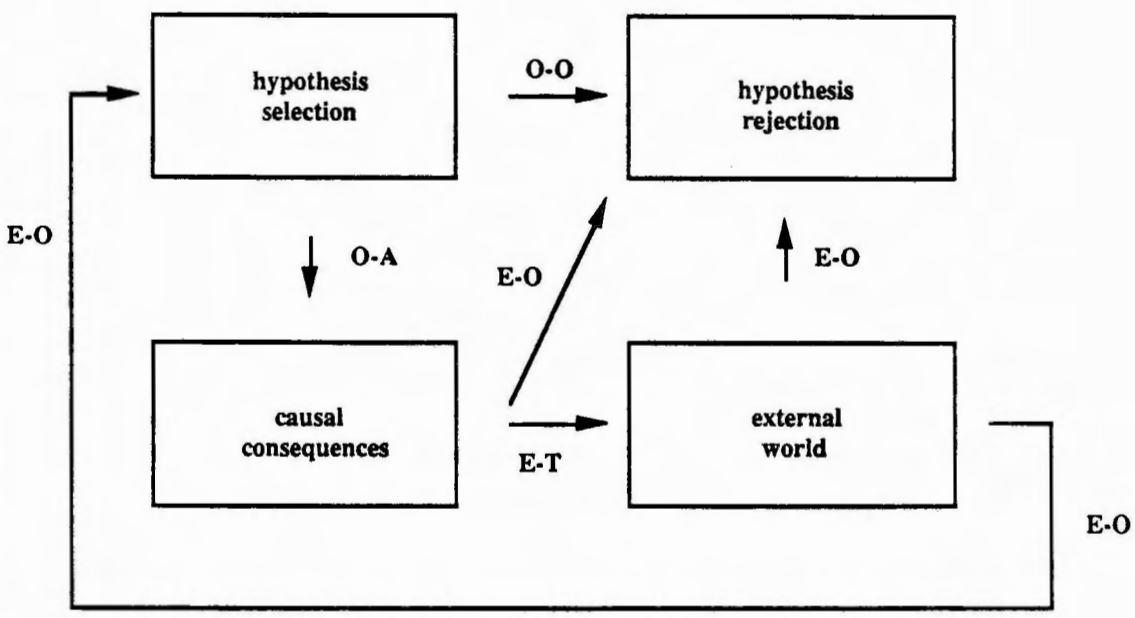


Fig 12 Hypothesis rejection based on causal knowledge

2. Downward assumption making interaction α_2 (type O-A)
The truth values of the output atoms **posshyp(h)** of the hypothesis selection are transformed into the same truth values of the input meta-facts **assumption(h)** of the causal consequences module.

3. Epistemic upward transformation α_3 (type E-O)
Here truth values of the output meta-facts of the form **true(s)** of the causal consequences module are transformed into the same truth values of the object level input atoms of the hypothesis rejection module of the form **cc_derived(s).**

4. Epistemic upward transformation α_4 (type E-O)
This interaction transforms truth values of the output meta-facts of the form **true(s)** of the external world module into the same truth values of the object level input atoms of the hypothesis rejection module of the form **observed(s).**

5. Epistemic-target meta-transformation α_5 (type E-T)
This transformation transfers truth values of the output meta-facts of the form **known(s)** of the causal consequences module into the same truth values of the input meta-facts of the external world module of the form **target(s, determine).**

6. Epistemic upward transformation α_6 (type E-O)
This interaction transforms truth values of the output meta-facts of the form **true(s)** of the external world module into the same truth values of the object level input atoms of the hypothesis selection module of the form **true(s).**

Of course, by this list of transformations not the whole dynamics of the reasoning pattern is specified. It is also necessary to specify control: what to do if the focus hypothesis is not rejected, and so on. We do not discuss this here. Also the possibility of adding an explicit observation selection module is not discussed.

4 Interaction types in learning systems

The interaction types as distinguished above are meant for knowledge-based systems that have fixed knowledge bases in their modules, i.e. systems that do not modify their knowledge bases themselves during reasoning sessions. Of course separate learning sessions might be possible to improve the performance of the system, but the framework as described above is not able to specify these sessions as reasoning processes in the sense as discussed above. However, it may have advantages if learning and reasoning in a system are integrated, for example in a system that permanently tries to improve its knowledge based on its experiences in reasoning. Therefore it may be useful to integrate possibilities for learning in our framework. The importance of such an integration was also pointed out to me by members of the machine learning community during the EKAW-90 workshop.

Learning, considered as self-improving activity, is a reflective capability of a system. In such an activity upward and downward reflections are needed to inspect the current knowledge

base (upward reflection) and to modify or extend the knowledge base (downward reflection). In the classification as discussed above we did not define interaction types for handling this type of reflective transformations. This is no essential omission: we simply did not try to cover learning systems. We sketch some possibilities to show how such interaction types can be added to the classification. As a simple example we have learning system in mind that consists of two modules. One of them is a reasoning module containing the knowledge the is being learned. The other one is the module that embodies the learning qualities. This second module (meta-module) reasons at a meta-level with respect to the first one (object module). Both modules can deal with cases. Cases consists of two parts: one part contains information on the attributes that are observable (observational information), and the second part contains information on the classification of the case (conclusive information).

The object module can use observational information of a case and may derive conclusive information on it by using its knowledge base. Whether or not this is successful depends on the quality of the current knowledge base. The meta-module uses as input both the observational and the conclusive information of a case. Besides, it can use meta-information on the object reasoning module on the current knowledge base, and on the outcome of the module for the case that is concerned (the latter information can be transformed by the already existing **E-O** interaction type). An upward interaction transforms this information from the object module to the meta-module. This enables the meta-module to inspect the current state and quality of the object module. The meta-module uses some form of (e.g. inductive) reasoning to draw conclusions on which modifications of the knowledge base are needed to improve its performance, e.g. which rules should be added to it. These conclusions are carried over to the object module by a downward transformation that carries out the proposed modifications. By this downward reflection the state of the object module (namely, its knowledge content) is changed.

To be able to specify the reflective interactions as sketched above, additional meta-predicates are needed. The most important one is the meta-predicate

rule(c, d)

specifying that there is a rule in the current knowledge base with **c** as a condition and **d** as a conclusion. Here **c** denotes a conjunction of literals and **d** a literal (i.e. an atom or the negation of an atom). By the upward transformation **kb_inspection** this meta-predicate can be trivially linked to the object level predicate **current_rule(c, d)** of the meta-module. The conclusions of the meta-module can be expressed using predicates **to_retract_rule(c, d)** and **to_insert_rule(c, d)**. The downward transformation **kb_modification** links truth values of these object predicates of the meta-module to truth values of the meta-predicate **rule(c, d)** of the object module in the following manner.

object-level outputs of meta-module	*truth value links*	*meta-level inputs of object module*
to_retract_rule(c, d)	**true ↔ false**	**rule(c, d)**
to_insert_rule(c, d)	**true ↔ true**	**rule(c, d)**

Using this downward transformation the object module can be modified. This gives rise to two new interaction types: **K-O** (knowledge base inspection) and **O-K** (knowledge base

modification). Extending our classification by these interaction types enables us to incorporate permanent learning integrated with reasoning.

5 Conclusions

One of the advantages of the logical foundation related to DESIRE is that it enables one to define in precise terms a generic part of the structure (formalized generic task model) which is independent of the specific domain knowledge that is involved in an application. This encourages reuse of (parts of) this generic structure (see [4]).

The interaction types as treated in this paper show us some of the possibilities for dynamic behaviour of a system that arise if a suitable modular structure is chosen. It turns out that to realize dynamic behaviour there is no need to give up a well-defined declarative semantics of the system. What is important though is to do some work to find the right type of conceptualization and formalisation of the (expert) task at hand. Once this work has been carried out one may take advantage of the clear declarative semantics that has been obtained; this concerns documentation of the design as well as implementation and maintenance.

After interactions between modules have been defined more formally it becomes possible to identify and classify a number of interaction types that are often used. Building a library of these interaction types may support the design, specification and implementation of modular knowledge-based systems. These interaction types may serve as building blocks. This may be compared to the approach of Chandrasekaran et al. (for instance, [1]).

This paper contains a number of examples and issues that only could be sketched. In some other papers these are or will be worked out in more detail.

Acknowledgements

Earlier drafts of this paper have been read and commented upon by Izak van Langevelde en Albert Philipsen. This has lead to a number of improvements in the text.

References

1. B. Chandrasekaran, Generic tasks in knowledge-based reasoning: high level building blocks for expert system design, IEEE Expert, Fall 1986
2. W.J. Clancey, Heuristic classification,
   Artificial Intelligence 27 (1985), p 289-350
3. G. Dam, A design for the strategic interactive medical diagnostic expert system SIMEDES,
   Report PE8901, University of Amsterdam, Programming Research Group, Expert Systems Section, 1989
4. W. Kowalczyk, J. Treur, On the use of a formalized generic task model in knowledge acquisition,
   Proc. 4th European Knowledge Acquisition Workshop, IOS Press, 1990, pp. 198-221
5. K.R. Popper, Conjectures and refutations, Routledge and Kegan Paul, London, 1963
6. Y.H. Tan, J. Treur, A logical description of a bimodular system for nonmonotonic reasoning with dynamic assumptions, Report P8826, University of Amsterdam, Programming Research Group, 1988
7. J. Treur, On the use of reflection principles in modelling complex reasoning,
   Intern. Journal of Intelligent Systems 5 (1990)

8. J. Treur, Design of modular and interactive knowledge-based systems,
Report P8826, University of Amsterdam, Programming Research Group, Expert Systems Section, 1988
9. J. Treur, A logical analysis of design tasks for expert systems,
Int. Journal of Expert Systems 2 (1989)
10. J. Treur, Incompleteness in domains with a layered empirical basis and heuristic reasoning in SIX,
Report PE8903, University of Amsterdam, Programming Research Group, Expert Systems Section, 1989
11. J. Treur, Modelling non-classical reasoning patterns by interacting reasoning modules,
Report IR-236, Vrije Universiteit, Department of Mathematics and Computer Science, 1990, to appear
12. R.W. Weyhrauch, Prolomenega to a theory of mechanised formal reasoning,
Artificial Intelligence 13 (1980), p 133-170

Fifth European Knowledge Acquisition for Knowledge-Based Systems Workshop

EKAW91

Knowledge Acquisition Tools

Case-Based Knowledge Acquisition, Learning and Problem Solving for Diagnostic Real World Tasks[1]

Klaus-Dieter Althoff & Stefan Weß

University of Kaiserslautern
Dept. of Computer Science
P.O. Box 3049, D-6750 Kaiserslautern
Federal Republic of Germany
e-mail: althoff(wess)@informatik.uni-kl.de

Abstract

Within this paper we focus on both the solution of real, complex problems using expert system technology and the acquisition of the necessary knowledge from a case-based reasoning point of view. The development of systems which can be applied to real world problems has to meet certain requirements. E.g., all available information sources have to be identified and utilized. Normally, this involves different types of knowledge for which several knowledge representation schemes are needed, because no scheme is equally natural for all sources. Facing empirical knowledge it is important to complement the use of manually compiled, statistic and otherwise induced knowledge by the exploitation of the intuitive understandability of case-based mechanisms. Thus, an integration of case-based and alternative knowledge acquisition and problem solving mechanisms is necessary. For this, the basis is to define the "role" which case-based inference can "play" within a knowledge acquisition workbench. We will discuss a concrete case-based architecture, which has been applied to technical diagnosis problems, and its integration into a knowledge acquisition workbench which includes compiled knowledge and explicit deep models, additionally.

1 Introduction

Within this paper we focus on both the solution of real, complex problems using expert system technology and the acquisition of the necessary knowledge from a case-based reasoning point of view. The development of systems which can be applied to real world problems has to meet certain requirements. E.g., all available information sources have to be identified and utilized. Normally, this involves different types of knowledge for which several knowledge representation schemes are needed, because no scheme is equally natural for all sources. Facing empirical knowledge it is important to complement the use of manually compiled, statistic and otherwise induced knowledge by the exploitation of the intuitive understandability of case-based mechanisms. Thus, an integration of case-based and alternative knowledge acquisition and problem solving mechanisms is necessary. For

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

this, the basis is to define the "role" which case-based inference can "play" within a knowledge acquisition workbench. Up to now all this problems are at most partially solved. What we need is a complete analysis of the problematic nature of knowledge integration for a complex real world domain. A first systematic approach is, e.g., given by van Someren et al. in [SZP90].

We will discuss a concrete case-based architecture (PATDEX: PATtern Directed EXpert System), which has been applied to technical diagnosis problems, and its integration into a knowledge acquisition workbench (MOLTKE: MOdels, Learning and Temporal Knowledge in Expert Systems for Technical Domains) which includes compiled knowledge and explicit deep models, additionally. For many of the just mentioned unsolved problems the proposed research work offers solutions, from an expert system point of view as well as from a case-based reasoning one. In connection with the first view these are the utilization of all available knowledge sources, a natural representation of the involved knowledge types, the exploitation of the intuitive understandability of case-based mechanisms and its application to diagnosis (including test selection). Research deficits concerning case-based reasoning are, e.g., the handling of real complex applications, the integration of case-based mechanisms with alternative ones for knowledge acquisition and problem solving, the integration of adaptive similarity measures and explicit deep models as well as the definition of a "role" for case-based reasoning.

PATDEX is based on a case representation (*diagnostic cases*) which is simple, natural and easy to acquire (a list of attribute-value-pairs). The applied inference engine can easily be understood because it is controled mainly by a similarity measure (together with two thresholds). Additionally, the classification task depends on the symptom relevances and the test selection task on the average diagnostic cost. The similarity measure is an instance of the *Contrast Model* proposed by [Tve77] and PATDEX acquires adaptively its analogical reasoning abilities for the support of the classification task. Thus, it can be applied to diagnostic problems in general[2]. PATDEX treats the test selection task itself in a case-based manner. This mechanism is based on *strategy cases* which can be automatically generated from the diagnostic cases. This offers the advantage to improve the classification and test selection strategy abilities separately from one another. PATDEX´s task within the workbench is the handling of exceptional cases which can be "temporary" or "absolute"[3]. Therefore PATDEX cooperates with the interpreter of the MOLTKE shell and utilizes the compiled and model-based[4] knowledge of the workbench as background knowledge. PATDEX is an interactive system and learns incrementally. If overgeneralizations are identified they can be corrected directly.

---

2 PATDEX combines two classification mechanisms, one for the solution process itself and one to acquire the necessary information while searching for a solution.

3 Exceptional cases are more seldom than typical ones. An exceptional case is called *absolute* if the described exception is related to the structure and range of the expertise. Such a case is called *temporary* if it refers just to the actual knowledge base and a change of its state may be probable.

4 In this case model denotes a deep functional model of the technical system which is to be diagnosed. It enables the simulation of the behavior of the technical system on a sufficiently precise level of abstraction.

PATDEX is able to handle complex real world applications. E.g., vague cases, default values for symptoms as well as pathologic symptoms and symptom values can be taken into consideration. Because of its efficiency and stability PATDEX can work, besides being integrated in the workbench, as a stand-alone system. PATDEX learns by forgetting and is able to dimension its case base explicitly. The efficiency of its procedure is ensured by a dynamical classification of the cases on different levels of activation as well as a Rete-like case network for the support of the inference engine. The latter one results in a fast exclusion of dissimilar cases, but guarantees that all really similar cases are considered. Thus, PATDEX offers solutions for many of the above mentioned problems which represent the current state of the art both in its combination and in some special aspects as the case-based test selection, the adaptive similarity measure and the integration in the workbench.

The next chapter gives an overview of PATDEX´s integration into the MOLTKE workbench while chapter 3 concentrates on the basic abilities of PATDEX as well as its cooperation with other workbench components. Chapter 4 offers different ways to evaluate and discuss PATDEX: two trade scenarios will be presented which give a distinct idea of the advantages and disadvantages of PATDEX. This is completed by a classification and discussion of PATDEX concerning related research work.

2 Integration of PATDEX into the Workbench

Technical diagnosis usually needs three main sources of information which have to be represented adequately. These are the compiled diagnostic knowledge, the empirical knowledge and the knowledge of the technical system which is to be diagnosed. The MOLTKE workbench allows the integration of all these knowledge types within one knowledge base [cf. Fig. 1 and 2].

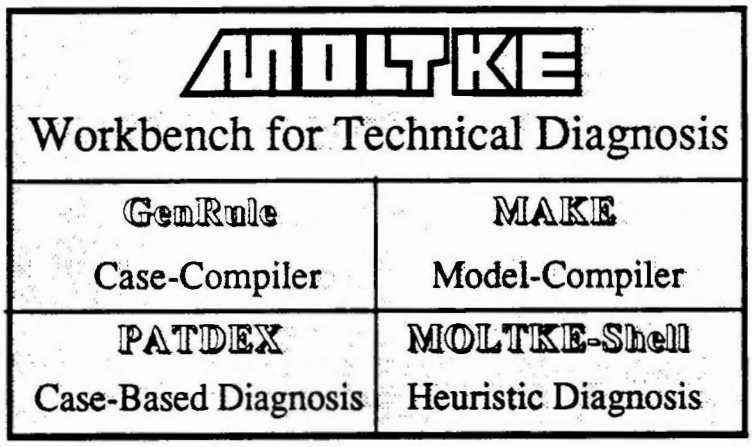


Fig. 1 – Components of the MOLTKE Workbench

The representation language of the workbench can be considered as a design model [BW89] and forms the basis for the MOLTKE shell (together with the interpreters). A *basic expert system* being automatically generated from the explicit model of the technical

system is the starting-point for the development of the knowledge base[5]. This can be extended and improved by the use of compiled and empirical knowledge. The generation and justification of different kinds of heuristic rules as well as the detection of certain inconsistencies within the knowledge base can be done automatically[6]. PATDEX´s task is the direct interpretation of the empirical knowledge which cannot, or at least not immediately, be integrated into the knowledge base. This kind of knowledge is called *absolute* or *temporary exceptional cases*. At any time there is the possibility of processing the empirical knowledge (exceptional cases) in parallel to the knowledge base. A procedure operating on the case memory for all cases decides which empirical knowledge can be integrated automatically. The actual state of the knowledge base is considered in a snapshot-like manner, but an automatic update to a changed knowledge base is possible[7]. All other cases are available for the PATDEX system in a special case base.

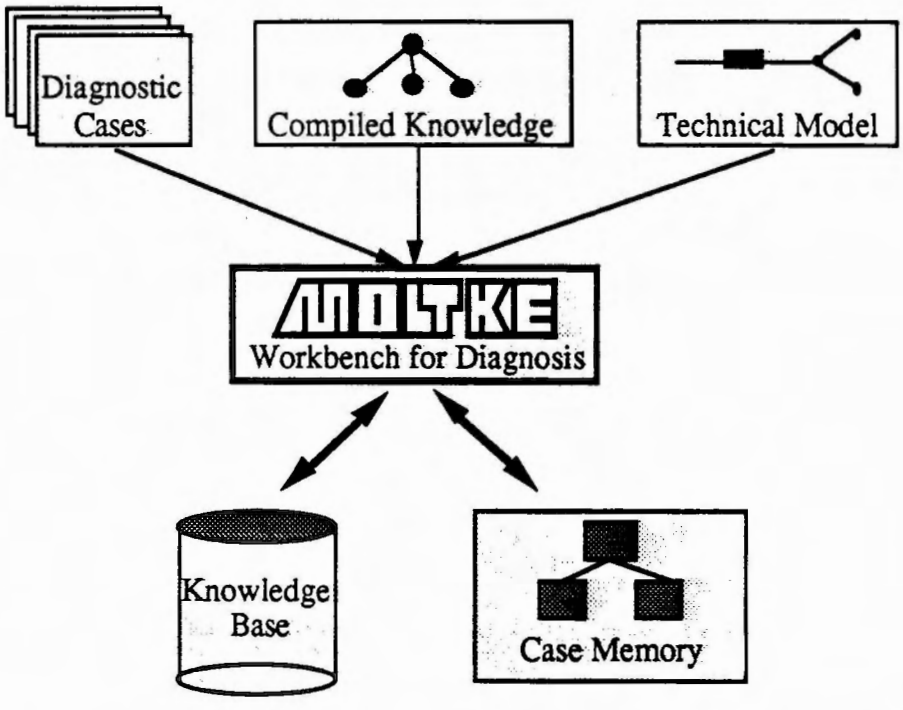


Fig. 2 – Processing of Multiple Knowledge Sources in the Workbench

3 The PATDEX System

In this chapter we describe the classification and the test selection components of PATDEX. They are completed by the description of the underlying similarity measure, the respective learning procedures, and the application of technical background knowledge. In addition, PATDEX´s cooperation with the other workbench components stated. For clarification purposes we begin with a short introduction into the needed terminology.

---

5 The basic expert system can be generated by the MAKE system [AMR90 or Reh91], cf. section 3.5.
6 This can be carried out by the GenRule system [AFT+89, AMR90, AMW90 or Alt91].
7 This is done by the KnowledgeBaseExpander, a subcomponent of the GenRule system [Alt91].

3.1 Terminology and Setting of the Task

We assume a fixed number N of *symptoms* $S_1,...,S_N$. With each symptom S_i a range R_i is associated; in principle symptoms are nothing than attributes. Typically R_i is either a real interval [a,b] or the boolean domain {0,1} or some other finite set. Symptoms may take on values in their range and these values are assumed to be the only source of information. Values of symptoms are obtained by carrying out a *test*. A test can be an observation, a measurement or simply the answer to a question. In some situations certain tests may not be allowed. The information at some stage of the diagnostic process is usually incomplete and is expressed in the form of an *information vector* or a *situation*: A situation is a vector $Sit = (a_{i1},...,a_{ik})$, $1 \leq i,j \leq N$ such that each $a_{ij} \in R_{ij}$. The components of Sit are the known symptom values whereas the values of the remaining symptoms are unknown. A situation is *complete* if every symptom has a value. A *diagnosis* (or fault description) is a formula of the first order predicate calculus using constants and relations over the ranges R_i; the precise form of these formulae is not of interest here. To avoid technical difficulties we assume always a single fault. This means that the set of complete situations is partitioned into sets representing these faults; a special set is "no fault" and, if wanted, another one is "unknown fault". The applicability of this approach relies on the fact that, at least, the "interesting" faults can be fully described. In the diagnosis of even complex machines this assumption is usually satisfied; in medical diagnostics this sometimes may be doubtful.

In a diagnostic problem some complete situations have occurred but are only partially known, i.e. one is confronted with some incomplete situation Sit. The task is to determine the diagnosis of the unknown complete situation (at least with some certainty).

3.1.1 Cases

A *(diagnostic) case* is the protocol of the real classification behavior of a service technician[8]. It is represented as a list of symptom-value-pairs (the problem description) completed by an empirical justified solution. In our diagnostic context the problem is a situation (Sit) and the solution is the diagnosis (Φ) justified by J, hence a case has the syntactic form $C = (Sit,J,\Phi)$. Since real world cases have to be handled three basic possibilities have to be considered:

- Sit did not have sufficiently many symptom values to determine Φ but one had a good guess.
- Sit did determine Φ but contained redundant information, i.e. unnecessary symptom values.
- Sit did determine Φ and no smaller situation would do so.

---

8 Service technicians are technicians of a manufacturer whose task is to maintain the functionality of sold technical systems.

Since one does not know which possibility has occurred PATDEX has to take care of all of them[9]. The (implicit) subject of all this considerations has been the classification task. For the test selection task we have a similar situation. A *strategy case* is a protocol of the real problem solving strategy of a service technician. The representation of strategy cases is comparable to that of the diagnostic cases whereas the solution is the symptom which is to ascertain next. Hence we have $C_S = (Sit,J,S_i)$.

3.2 Overall Algorithm of PATDEX

PATDEX's problem solving process is started by the user giving some observed symptom values as input to the system. Then the toplevel algorithm of PATDEX reads as follows (cf. also Fig. 3):

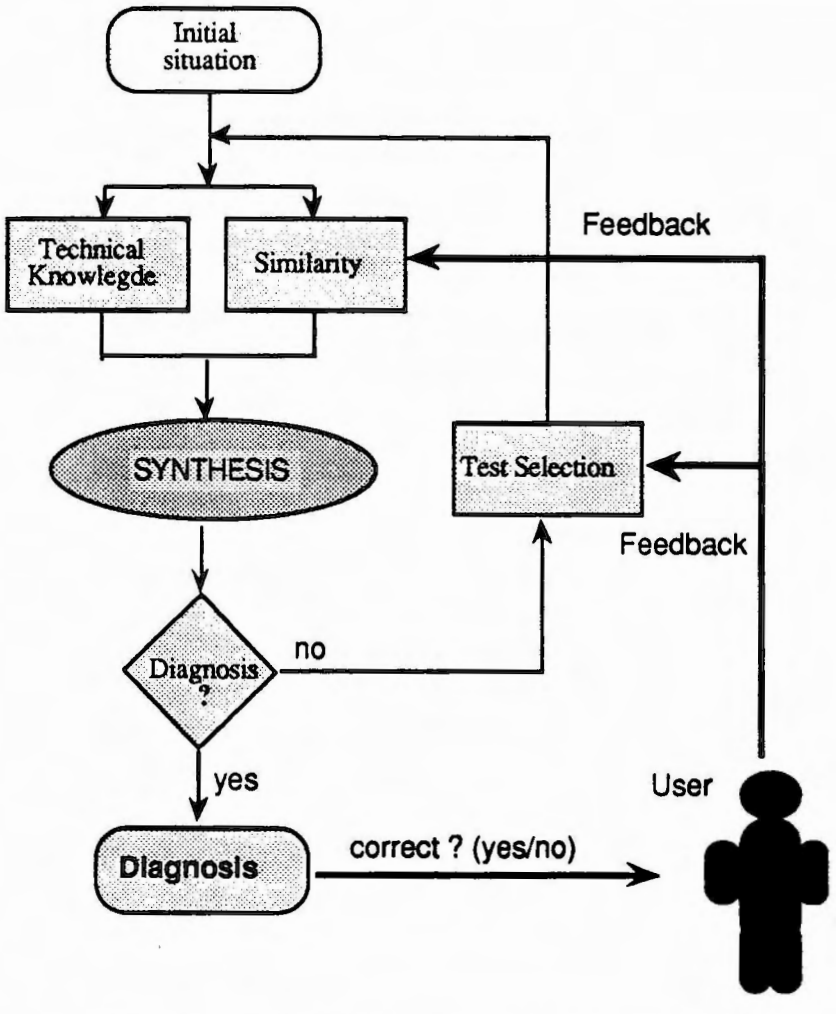


Fig. 3 – Toplevel Algorithm of PATDEX

9 Especially we point out here, that the occurrence of incompleteness (1) and redundancy (2) is independent of one another.

Input:	The actual situation Sit
Output:	Diagnosis Φ or negative report
1)	Find a case in the case base with a situation Sit' most similar to Sit. If there is no case with a situation at least "minimally similar" to Sit then stop with negative report.
2)	If Sit and Sit' are "sufficiently similar" then accept the diagnosis Φ of Sit' also for Sit and goto 4).
3)	Otherwise select a test in order to obtain an improved situation and goto 1).
4)	If the diagnosis is correct then add the case (Sit,J,Φ) to the case base and stop with success.
5)	If the diagnosis is not correct then cancel temporarily (i.e. for the actual problem) all cases with diagnosis Φ and goto 3).

Here we need an external teacher who says whether a diagnosis is correct or not. We also have to explain "minimally similar" and "sufficiently similar". For this we need a partition of the case base which is given after the introduction of the similarity measure. Finally, we have to describe the selection of the next test.

3.3 Solving the Classification Task

The goal of the classification task is to find a case which can be used to solve the current problem, i.e. to find a case which can explain the actual situation Sit correctly. Here we can exploit the characteristics of the domain of technical diagnosis that observable similarities, concerning the fault behavior of the technical system being considered, normally have similar causes. Thus, our underlying assumption is that similar cases have similar classifications. The selection of a suitable case uses a similarity measure *sim* which maps any combinations of situations into [0,1]. The basic axioms for *sim* are:

- $sim(x,x) = 1$ (reflexivity)
- $sim(x,y) = sim(y,x)$ (symmetry)

Given Sit, *sim* selects C´ = (Sit´,J,Φ) such that Sit´ is most similar to Sit with respect to the case base. If the current situation Sit is identical to Sit´ then Φ is the correct diagnosis for Sit. Often such an inference is still correct if Sit´ differs from Sit "a little bit", i.e. the two situations only have to be "sufficiently similar" with respect to the classification task.

3.3.1 A Static View on Measuring Similarity

A first approach to define "sufficiently similar" is to match the respective situations syntactically where the computed similarity has to exceed a certain threshold $\delta \in [0,1]$, i.e. $sim(Sit,Sit') > \delta$. A very general type of appropriate similarity measures for situations which are used in PATDEX is of the form:

$$sim(Sit_1, Sit_2) = \frac{\alpha\ card(E)}{\alpha\ card(E) + \beta\ card(D) + \eta\ card(U_1) + \gamma\ card(U_2)}$$

where α, β, γ and η are real numbers, card denotes the cardinality of sets, and

E	:=	set of symptoms with the same values for Sit_1 and Sit_2;
D	:=	set of symptoms with different values for Sit_1 and Sit_2;
U_1	:=	set of symptoms with values for Sit_1 but not for Sit_2;
U_2	:=	set of symptoms with values for Sit_2 but not for Sit_1.

PATDEX uses, as a first proposal, the parameters $\alpha = 1$, $\beta = 2$, $\gamma = \eta = 1/2$. This special choice of the parameters is mainly motivated by experimental results. The similarity measure has a defensive, pessimistic character. A high negative contribution to the measure is given for conflicting symptom values, i.e. we strongly wish to avoid false diagnoses.

A case becomes disqualified for further use in a particular diagnosis session as soon as all symptoms contained in that case do not hold, given a situation encountered during diagnosis, or if there are no unknown symptom values any more and the specified case does not exceed the diagnosis-threshold δ. Another reason for disqualification is given if the case the system chooses as its hypothesis is refused by the user. For performance purposes a second threshold ϵ is introduced with $0 < \epsilon < \delta < 1$. Both thresholds are locally defined for each case of the case base and enable a dynamic organization of all cases within different levels of activation. Depending on Sit_1 and Sit_2 these levels are called as follows:

- indistinguishable $\Leftrightarrow$ $sim(Sit_1, Sit_2) = 1$;
- sufficiently similar $\Leftrightarrow$ $\delta \leq sim(Sit_1, Sit_2) < 1$;
- probably similar $\Leftrightarrow$ $\epsilon \leq sim(Sit_1, Sit_2) < \delta$;
- at least minimally similar $\Leftrightarrow$ $0 \leq sim(Sit_1, Sit_2) < \epsilon$;
- not minimally similar $\Leftrightarrow$ $sim(Sit_1, Sit_2) < 0$.

3.3.2 Extending the Static View

Within case-based reasoning retrieval can be called "reasonable" only if the solution of the retrieved case can be applied to the current problem. Thus, a diagnostic case is sufficiently similar to a given situation if they have the same diagnosis. The problem with this definition of similarity is that the diagnosis of the given situation is not known at retrieval time (of course). Therefore we have to find a good approximation for this measure. The complexity of the knowledge which is necessary to define a well-suited static similarity measure is comparable to the complexity of the whole diagnostic task. Thus, the definition of a similarity measure for a special domain has to be seen as a learning task.

This extension results in an improved version of the similarity measure, but still does not consider any explicit domain knowledge. Within the domain of technical diagnosis this leads to the following problems:

- the use of compiled diagnostic knowledge
- identification of redundant information
- the handling of pathologic symptom values
- similarity between symptom values

The use of compiled diagnostic knowledge

Real diagnostic situations are characterized by an incomplete information vector, i.e. there exist some symptoms of which the value is unknown. Technical knowledge can be used to derive such values from known ones. E.g., from "light = on" it can be derived that "switch = closed" holds with a certain probability. Here we can differentiate between causal and heuristic relations. In the MOLTKE workbench both types of knowledge can be automatically generated by specialized components. Thus, this information is utilized by the PATDEX system.

Identification of redundant information

The relevance of a symptom for a certain diagnosis depends on the respective context. An important aspect of our PATDEX approach is to view the relevances of certain symptom values for special situations as a part of the empirical knowledge which shall be learned. In PATDEX we combine the case-based reasoning approach for diagnosis with a connectionist approach for learning this empirical knowledge. These relevances $w_{ij} \in [0,1]$ are represented by means of a *relevance matrix* $\boldsymbol{R} = [w_{ij}]$ where the symptoms S_i and diagnoses Φ_j occur as inscriptions of the rows and columns, respectively. In course of time the weights of the symptoms, i.e. the elements of the relevance matrix, are learned by PATDEX. The strategy for learning the entries of the relevance matrix is similar to the competitive learning mechanism proposed in [RZ85].

The handling of pathologic symptom values

We can differentiate between symptom values which are pathologic or not. Pathologic symptom values are caused by an abnormal system behavior while non-pathologic values reflect the normal functioning of the technical system. For the degree of relevance of a certain symptom value it is important whether it is a consequence of the normal functioning of the technical system or of a fault. E.g., "relais 21K3 switched" is of the first kind while "voltage 214 too high" is a pathological symptom value. The differentiation whether a certain symptom has a pathologic character or not depends on the respective context, e.g. "light≠on" is not automatically a pathologic one. If "switch=closed" is known additionally, the previous symptom becomes pathologic. Thus, functional background knowledge is needed for the handling of such information. Within the PATDEX system every most similar case has to explain all known pathologic symptom values. This defines a kind of "minimal correctness".

Similarity between symptoms

Every really flexible similarity match between the current situation and a case of the case base has to consider the similarity between symptoms and symptom values (Similarity versus Part-Identity [Smi89]). This can be realized by the additional use of local similarity measures $\omega_i(a_{ik},a_{il}) \in [0,1]$, which determine the similarity between possible symptom values $a_{ik},a_{il} \in R_i$ of a symptom S_i, or by a knowledge-based inference (i.e. $\omega_i(a_{ik},a_{il}) \in \{0,1\}$). If one of the symptom values is *unknown* then the similarity ω_i evaluates to 0.

The introduction of $\boldsymbol{R}$ and ω_i leads to the following extended definition of the similarity measure *sim*:

$$sim(Sit_1, Sit_2) = \frac{\alpha E}{(\alpha E + \beta D + \eta U_1 + \gamma U_2)}$$

For the exploitation of the above described features we have to extend the definition of the "attribute sets"[10]. Let a_{ik} be the observed symptom value in Sit_1 and a_{il} the defined symptom value in the actual case $C = (Sit_2, \Phi j)$. Then the new definition reads as follows:

$$\boldsymbol{E} := \sum_{S_i \in E} w_{ij}\ \omega_i(a_{ik}, a_{il})$$

$$\boldsymbol{D} = \sum_{S_i \in D} w_{ij}\ (1-\omega_i(a_{ik}, a_{il}))$$

$$\boldsymbol{U_1} := \sum_{S_i \in U_1} v_{ij}\,(1-\omega_i(a_{ik}, a_{il})) = card(U_1) \text{ with } S_i \text{ pathologic}$$

$$\boldsymbol{U_2} := \sum_{S_i \in U_2} w_{ij}\,(1-\omega_i(a_{ik}, a_{il})) = \sum_{S_i \in U_2} w_{ij}$$

We point out here that ω_i is zero for symptoms S_i which belong to one of the attribute sets U_1 or U_2, because the corresponding symptom values are unknown. Additionally, we restrict the representation of redundant symptoms (i.e. $S_i \in U_1$) to pathological ones. Thus, observed redundant symptom values representing the normal behavior of the underlying technical system cannot decrease the value of *sim* any more. Since PATDEX focuses on the learning of symptom relevances only for the respective diagnosis no entries for redundant symptoms S_i can be created. Here we need an alternative weighting v_{ij}. In PATDEX we define $\forall i,j\ v_{ij} = 1$, which is motivated by the above mentioned restriction of U_1.

By the use of these definitions we get a similarity measure *sim* which is depending on the values represented in the relevance matrix. After each erroneous diagnosis the weights of the relevance matrix are changed. Thus, the similarity measure *sim* is the result of an adaptive learning process and the integration of background knowledge available from the MOLTKE workbench.

3.4 Solving the Test Selection Task

For a case-based reasoning system not only its correctness is of importance but also its efficieny. The task of the test selection component is to complete a given incomplete situation as economically as possible, i.e. to avoid redundant tests, such that a diagnosis can be made. Up to now existing case-based reasoning systems concentrate on the classifica-

[10] Here we replace the set cardinalities by the sum of the symptom relevances.

tion task only. The test selection task either is not considered[11] or realized in very simple manner. If cases are acquired for the classification task it is on offer to use them for the test selection task, too. In PATDEX the diagnostic cases reflect the whole diagnostic process. Thus, not only the classification process is described but also that of test selection. Therefore the diagnostic cases have to represent the strategy of the respective service technician, i.e. within the case representation we have to assume a temporal order.

For a stepwise optimization of the classification abilities the user´s feedback can be utilized. A correct diagnosis results in a positive feedback, a wrong one in a negative feedback. Here it is clear whether a diagnosis is correct or not. Within the test selection context such a clear definition is not possible, because there do not exist any absolute criterions whether the ascertaining of a symptom is necessary or not. Thus, for the test selection task criterions like fault frequency and diagnostic cost, i.e. the cost which is necessary to ascertain a (special) symptom, come into play which have to be taken under consideration for the feedback. A possible approach for learning a good test selection strategy is to adaptively form a test selection order which is not necessarily optimal for a special case but for the average of all known cases. Of course, it has to be taken into account that special or exceptional strategies normally are more informative than typical ones. Thus, in reality a combination of a case focusing and an average cost minimizing strategy is the best suited approach which is possible.

Of importance for the feedback is the notion of *subsequent cost*. Subsequent cost denotes the cost which still remains to find the correct diagnosis after of a certain strategy case has been applied. E.g., a simple measure for this is the number of symptoms which still have been acquired after a strategy case has been applied. More sensible are domain-specific measures which, additionally, consider diagnostic and temporal cost.

Algorithm for case-based test selection:

Input:	Diagnostic case C_k (diagnosis hypothesis), Situation Sit
Output:	Symptom S_i which is to ascertain
1)	Find a strategy case in the case base with a situation Sit' most similar to Sit.
2)	If there is no strategy case with a situation at least "minimally similar" to Sit then use diagnostic case C_k for a case focusing test selection.
3)	If several strategy cases are "sufficiently similar" then select that strategy case which has caused the lowest subsequent cost on the average.

After a diagnostic session has finished every strategy case is charged with the diagnostic cost it has caused. Strategy cases which encompass redundant symptoms are charged, on the average, with higher cost than cases which include only necessary symptoms. The fault frequences are taken into account implicitly by the computation of the averages. If the system has carried out an unknown strategy then the respective strategy cases are included into the strategy case base. In the worst case all symptom orders have to be represented by strategy cases. Thus, the needed space increases exponentially concerning the

[11] This is based on the unrealistic assumption that all needed information is given.

number of symptoms. This is not possible for a real world application. In PATDEX the number of representable strategy cases is fixed (e.g. 10000). If more cases have to be represented PATDEX forgets those strategy cases which have not been used for the longest period of time.

The idea behind the case-based test selection of PATDEX is to offer good strategies for frequently occuring situations. Neither the frequency of a fault nor the quality the strategy taken by the system have to be defined a priori. For faults which appear less frequently PATDEX can utilize the strategy of the service technician which is represented by the actual (diagnostic) case, i.e. a case-focusing strategy can be chosen. Of course, such a strategy is not necessarily globally optimal, but exceptional cases normally are more informative than typical ones.

3.5 Cooperation with other Components of the Workbench

Since we have given an overview of the MOLTKE workbench and described PATDEX in more detail we are now able to summarize the cooperation of PATDEX with the other components of the workbench, especially the MOLTKE shell:

- **Cooperation with GenRule** (Generator of Empirical MOLTKE Rules)
  GenRule determines, and regularly updates, the subset of the overall case memory which is available in the PATDEX case base. PATDEX uses heuristic rules generated by GenRule for the derivation of additional symptom values (from already known values).
- **Cooperation with MAKE** (Model-based Automatic Knowledge Extractor)
  PATDEX works in parallel to the basic expert system generated by MAKE which is the first version of the knowledge base. MAKE generates causal rules which are used by PATDEX, in a similar way as the heuristic rules of GenRule. In addition, the MAKE underlying model serves data base-like queries, e.g. for the identification of concrete components of the technical system.
- **Cooperation with the MOLTKE-Shell**
  The MOLTKE-Shell controls the test selection while PATDEX does its case-based classification as a background job. The test selection component of PATDEX is used for clarification purposes only, i.e. if contradictory classifications of the systems are present and cannot be solved immediately.
  In view of the procedure the following situations are of special interest:
  - The shell classifies correctly and PATDEX does not find a sufficiently similar case in its case base:
    - » The diagnosis can be made, because no contradiction has been observed.
  - Both the shell and PATDEX classify correctly:
    - » The diagnosis can be made. There is the possibility to exclude the cases from the case base, because the knowledge base already "encompasses" them.
  - PATDEX classifies correctly and the shell does not find any diagnosis:
    - » There is a gap in the knowledge base. If the most similar case within the PATDEX case base is a temporary exception, then the knowledge base can be extended and the case excluded from the case base.
  - The shell and PATDEX classify differently:

» Different conclusions have to be taken depending on the actual situation. It is possible that PATDEX is overgeneralizing and the shell is classifying correctly, or that PATDEX is classifying in a more special way than the shell and an exception can be identified. It is necessary to prove the case chosen by PATDEX. If the case is proven to be sound, then the knowledge base does not fit to the current situation. It has to be checked if the knowledge base still makes sense taking a more general point of view. If it is not possible to prove the selected case the respective diagnosis threshold has to be increased.

- Other situations, e.g. wrong classifications by PATDEX or the shell:
  - » The correct case can be directly given as input. Afterwards PATDEX can correctly classify the identical situation. If necessary wrong parts of the knowledge base have to be corrected.

- **Case-based test selection within the shell:**
  PATDEX's test selection mechanism is a component of the shell.

4 Evaluation and Discussion

To enable an intuitive evaluation of the PATDEX system two scenarios are presented[12]. The basic idea is that a software developing company *ExpertSystemsDeveloper (ESD)* tries to sell the PATDEX™ system and/or the complete MOLTKE™ workbench to a company named *TechnicalSystemsManufacturer (TSM)* which might need them for the solution of their diagnostic problems. *TSM* makes certain demands on the application of such tools. It takes the advice of *ESD* which, additionally, makes certain offers. The subject of the first scenario is PATDEX as a stand-alone system while in the second one PATDEX is only a part of the subject under consideration, namely the MOLTKE workbench. These scenarios are completed with two evaluations, one given by *TSM* the other one by *ESD*. Then PATDEX is classified and discussed and the current state of realization of all involved workbench components is clarified.

4.1 Consultation of the Manufacturer

TSM wants to use expert systems for the support of its service department. After a thorough analysis of *TSM*'s current situation *ESD* makes an offer which, among others, includes the following items:

- **Possible locations of application:**
  - Usage at the manufacturer
  - Centralized application at a client of *TSM*
  - Decentralized application at a client of *TSM*
  - Local usage within a technical system itself
- **Possible types of application:**
  - Support of telephone diagnosis
  - Usage as integrated and/or accompanying diagnostic system
  - Support of training
  - Support of knowledge documentation

12 Of course, these scenarios are fictitious, though correspondences with real situations are intended.

TSM responds affirmatively to ESD's offer. It decides to support the service of the machines of type MX by the use of a diagnostic expert system. At the same time this should be seen as a test case for the efficiency of ESD's products. Then ESD gives an introduction into the problematic nature of developing a knowledge base for a technical system like MX. ESD illustrates this using six different sets of cases which, starting with the smallest set A, are subsets of the respective following set [cf. Fig. 4]:

- **Set of all input cases (A)**
  This are all cases which have been given as input to the knowledge base.
- **Set of all cases which can be correctly classified (B)**
  This set encompasses all cases of which the diagnosis would be selected if the problem description of the respective case is given. If identity is viewed as the best similarity being possible, as done by PATDEX, then all input cases can be correctly classified which results in $A \subset B$.
- **Set of all interesting cases (C)**
  Set C includes all cases which must be correctly classified according to the meaning of the responsible experts. Since only real cases should be given as input and all correctly classified cases are, of course, interesting C is a superset of B: $B \subset C$.

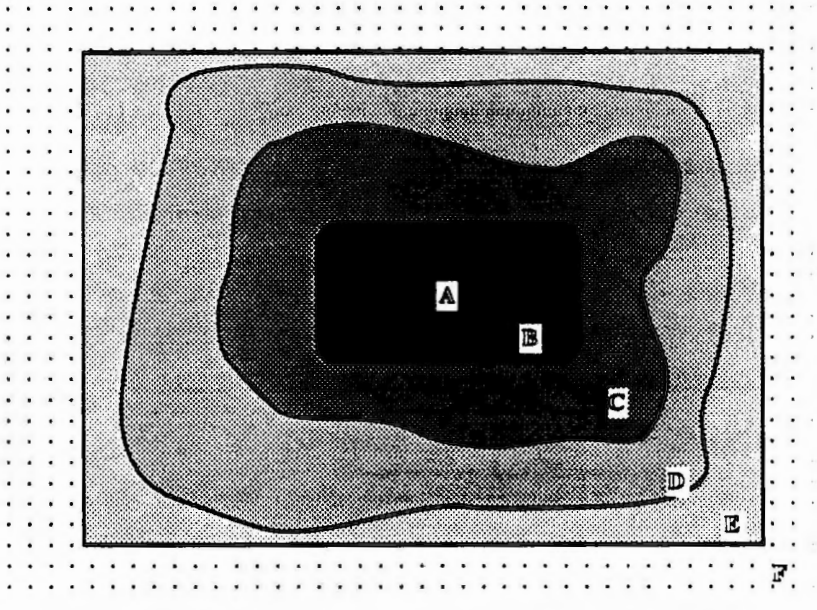


Fig. 4 – Case Scenario for Diagnostic Real World Applications

- **Set of all cases which are possible in view of the functional behavior of the technical system and the most precise level of abstraction which is still being considered (D)**
  This are all cases which are really possible though they would not be considered as interesting (if they do not belong to set C), because the experts do not remember (or know) them, i.e. the experts would not present them as input or test cases[13].

[13] For clarification purposes we want to mention here that *test cases* have to be distinguished from *strategy cases*. While the latter ones support the test selection task, i.e. the ascertaining of symptoms, test cases are a means for carrying out a test of the expert system.

Since all cases being interesting for the experts are possible according to the functional behavior of the technical system, this results in $C \subset D$.

- **Set of all cases which can be described by the chosen case representation (E)**
  Set E encompasses many cases which do not appear in reality because of technical constraints of the system which is to be diagnosed. The case representation is aimed at being able to describe all cases which belong to D. Thus, we have $D \subset E$.
- **Set of all really possible cases facing the technical system under consideration (F)**
  This set includes all really possible cases according to the behavior of the technical system, especially those of which the level of detail of the case description is more precise than it is necessary for the diagnosis which is to be made. Thus, many of these cases cannot be described using the chosen case representation. Another aspect is that all cases are included which ignore the technical constraints of the underlying system. This results in $E \subset F$.

ESD calls *TSM*'s attention to the fact that PATDEX correctly classifies all cases which have been given as input to it. *TSM* refers to the point that all interesting cases have to be diagnosed correctly. *ESD* replies that this would be the case, if all interesting cases are given as input to the system. *TSM* rejects this because of the unjustifiable expenditure and guarantees only a restricted selection of exemplary cases to *ESD*. In addition, *TSM* insists on the carrying out of several exhaustive tests which would be very important concerning the acceptance of the expert system.

4.2 Scenario 1: PATDEX as a Stand-Alone System

TSM's guideline for the development of a MX knowledge base can be roughly described by the following quotation: "It is better to solve only 70% of the cases with low cost than all cases with high cost". Therefore *ESD* decides to develop the diagnostic expert system for the MX only on the basis of PATDEX.

The development of the knowledge base is subdivided into several steps. Within step 1 cases are defined as protocols of the real problem solving behavior of a service technician and represented in PATDEX. Then these cases are acquired by "observing" the service technicians. To reduce the observation expenditure a symptom acquisition blank has been designed which can be used by the service technicians to mark and/or enter the symptom values that have been ascertained during the diagnostic process.

During step 2 an overall test of the knowledge base which has been acquired in step 1 is carried out. *TSM* engages different experts who have to check the knowledge base by the use of many test cases being selected for that very purpose. Because of the big gaps within the knowledge base and some wrong diagnoses the tests results in a negative evaluation.

ESD refers to two facts, namely that within step 1 input cases have been available only in an insufficient number and that step 2 (as well as step 3) have to be seen as a part of the knowledge acquisition process. In the meantime PATDEX would be able to

correctly classify all test cases which have been given as input to it, i.e. gaps of the knowledge base can be eliminated easily. In addition, tests could be seen as an appropriate procedure for the explication of the only implicit available cases of set C. Upon request all involved TSM-experts have to admit that they remembered many of the applied test cases only during the testing of the knowledge base. Moreover, it has been very motivating to search for situations which PATDEX cannot handle. The wrong diagnoses made by PATDEX can, to a big part, be traced back to overgeneralizations. ESD underlines the fact that every diagnosis, where PATDEX should be in principle able to make it, must be a part of at least one case of the case base. Wrong diagnoses which cannot be traced back to overgeneralizations can be eliminated by PATDEX´s learning component (adaptive learning of the symptom-diagnosis-relevances) or manually because of the transparency of the applied case-based inference.

Thus, the results of step 2 meet, in spite of the negative intermediate evaluations, the requirements of TSM. The main criterion for the acceptability of the solution offered by PATDEX has been the scalability of the validity of the knowledge base which easily enables the fulfilment of TSM´s guideline. TSM decides to use PATDEX within a third step for a special adaptation process where only the real cases of the respective MX machine (one several selected ones) are given as additional input to the expert system.

4.3 Scenario 2: PATDEX as a Component of the Workbench

Since the MX project has been successful TSM takes further diagnostic expert system projects into consideration. Especially the problematic nature of determining set C has been realized by TSM only now. Besides, the use of expert systems for the support of training and knowledge documentation recommends the development of knowledge bases which are as complete as possible, even then if the cost for it is very high. Some critical TSM-cooperators refer to the maintenance and update problem of such high cost software systems. ESD suggests to analyse which additional information source can be utilized. The result is that two further types of knowledge sources can be exploited, namely compiled diagnostic knowledge and, on a large scale, the technical documentation of the particular machines.

Then ESD makes the offer to develop a knowledge base on the basis of the whole MOLTKE workbench. as for the MX project this project is aimed at modeling TSM´s knowledge about a special type of machine which in this case has to be as complete as possible. Facing the case sets of Fig. 4 ESD explains that such a project has two consequences. First, the equality of the sets B and C (as in the MX project) and, additionally, the equality of the sets C and D. Therefore it would be much more difficult to determine set C. ESD calls TSM´s attention to the fact that the only procedure which makes sense is to represent as many parts of the technical system as possible as a deep explicit model, and, thus, make it available for the expert system. This must be completed with very detailed and exhaustive test steps which enable additional manual improvement of the knowledge base.

TSM responds affirmatively to ESD´s offer. TSM holds out the prospect of another project which deals with the fault diagnosis of the simpler structured MX variant MY. It

would be possible to develop a complete, new knowledge base and to do this in very efficient manner, because most of the knowledge being acquired for the MX can be transfered to the MY knowledge base. In addition, this knowledge base is not as extensive as that of MX. The basic condition for starting the project is that there must exist an efficient strategy how to update an existing knowledge base and how to adapt it for comparable technical systems.

For clarification purposes ESD gives additional advice concerning the MOLTKE workbench (cf., e.g., Fig. 1 and 2). During the training sessions TSM-cooperators are especially interested in the cooperation between PATDEX (which they already know) and the other components of the workbench (cf. section 3.5). ESD explains that a basic version of the knowledge base, which is to be developed, can be automatically generated from the technical documentation which has been given as input to the workbench. Then this version has to be manually revised and extended. E.g. one problem which has to be solved is that this basic system does not simulate the real experts´ problem solving behavior. This problem can be partially eliminated by the automatic generation of heuristic rules. Within the workbench PATDEX´s main task is the direct interpretation of the available concrete empirical knowledge. This ensures that as little empirical knowledge as possible is not or in a wrong way represented in the knowledge base. Thus, cases are used for the refinement and correction of the knowledge base as well as for the orientation of the test selection task to the expert´s problem solving behavior. All this is aimed at the development of an integrated general knowledge base which can be processed using the MOLTKE shell interpreter. Exceptional cases stay in the PATDEX case base and are still processed by PATDEX. The shell and the PATDEX system are cooperating dynamically.

One main subject of the training sessions was the maintainability of a developed overall knowledge base (which encompasses the knowledge base and the case base). Minor changes which are needed variants and/or new series of a technical system can be handled without problem using PATDEX. Major changes can be carried out directly in the knowledge base or in the underlying deep model of the technical system. If the latter alternative is chosen a completely new version of the basic knowledge base has to be generated. In addition, all manual changes which have already been done have to be repeated. This produces a certain dissonance among the TSM-cooperators because of the impression that (nearly) the same work has to be done very often. ESD refers to the fact that this would be in principle a software development problem. Therefore ESD cooperates with a research group which currently improves the workbench. This relates to the development of a component for knowledge maintenance and consistency checking, the incremental extendability of the basic version of the knowledge base as well as adaptation of the knowledge base to given changes of the underlying deep model and the structural adaptation of the knowledge base to known cases. Because of the positive experiences with the PATDEX system TSM at last decides to carry out the MY project as another test case.

4.4 Related Work

In view of the classification task the analogy-based inference mechanism of PATDEX can be considered as "transformational analogy", facing the test selection task it can be seen

as "derivational analogy" [Car83+86]. In addition, there exist several strong connections to the "Memory-Based Reasoning" approach of Stanfill & Waltz [SW88] and the "Instance-Based Learning" (IBL) algorithms proposed by [AKA91].

Well-known case-based reasoning approaches which can be compared to PATDEX are, among others, the PROTOS [Bar89], CASEY [Kot89], SIZZLE [Off89], CREEK [Aam90], and memory-based reasoning, e.g. MBRtalk [SW88] approaches. Good overviews are given in [KRW89] and [Sla91]. In applying case- and analogy-based reasoning techniques to the problematic nature of test selection and in integrating connectionist methods for adaptive learning, PATDEX contributes to the current state of the art defined by these (and similar) systems, respectively. PATDEX has its own pragmatic view on case representation and similarity [RW91] like SIZZLE. It does not use a case memory to represent the similarities between cases like, e.g., [VC89], but exploits an explicit evaluation function in combination with a partitioning of the case base called levels of activation and a Rete-like case network. This is comparable to the indexing techniques in conceptual memories cf., e.g., [Kol83a+83b]. Since real world applications involve large case bases the retrieval must be very efficient. Our approach is to combine the good performance of a syntactic match like MBRtalk, or the IBL algorithms, with, if necessary, the (expensive) use of explicit background knowledge like, e.g., CREEK and CASEY do. This approach is possible because there exists a defined role for the PATDEX system within the MOLTKE workbench, whereas CREEK and CASEY have to do the whole diagnostic task stand-alone.

PATDEX not only exploits causal background knowledge, as CASEY does, but also functional knowledge. PATDEX is a computational model which is fully implemented. Especially PATDEX/1 [cf. section 4.5] has been tested since early 1989. Because of its adaptive capabilities PATDEX can be easily used for other domains within the field of technical diagnosis. This is not so easily done for PROTOS as many numerical values and the relational structure needed for the explanation facility have to be transferred to the new domain [BBP88].

In the MOLTKE workbench the applied learning strategies have to support the generation of an implementation model for the underlying technical system, based on a design model for technical diagnosis. As compared with, e.g., the BLIP approach [Mor87+89b] this is a simplifying view because no domain model is learned, but we think that our approach is adequate for the field of technical diagnosis.

4.5 State of Realization

Actually, there are two systems, PATDEX/1 and PATDEX/2. PATDEX/2 [Alt91, Weß91] is fully implemented and integrated into the MOLTKE workbench [Ric91]. PATDEX/1 as a stand-alone prototype is already available since early 1989 [AKW+89, AMW+89]. The MOLTKE shell [AMR90], the GenRule system [AMW+90], [Alt91], and the MAKE system [Reh91] are fully implemented, too.

5 Conclusion

PATDEX has been designed for real world applications in the field of technical diagnosis and has its well-defined role within the MOLTKE workbench. PATDEX is fully integrated in an overall view of knowledge acquisition and integration as well as learning. Having this view in mind PATDEX contributes to the current state of the art for case-based reasoning systems.

6 Acknowledgement

Thanks go to especially to Prof. Dr. Michael M. Richter, Frank Maurer, Mike Stadler and Ralph Traphöner and our machine learning research group at Kaiserslautern, for many very engaged discussions which have been an important step for the development of the PATDEX systems as they are described here.

References

[Aam90] Aamodt, A. A Computational Model of Knowledge-Intensive Learning and Problem Solving. In: [WBG+90], pp 1-20

[AFT+89] Althoff K-D, Faupel B, Traphöner R et al. Knowledge Acquisition in the Domain of CNC Machining Centers: the MOLTKE Approach. In: *Proc. EKAW-89*, pp 180-195

[AKA91] Aha, DW, Kibler, D, Albert, MK. Instance-Based Learning Algorithms. *Machine Learning* 1991, 6: 37-66

[AKW+89] Althoff K-D, Kockskämper S, Weß S et al. Ein System zur fallbasierten Wissensverarbeitung in technischen Diagnosesituationen. In: Retti J & Leidlmeier K (eds), *5. Österreichische Artificial-Intelligence-Tagung*, Springer Berlin Heidelberg New York, 1989, pp 65-70

[Alt91] Althoff K-D. *Eine fallbasierte Lernkomponente als integrierter Bestandteil der MOLTKE-Werkbank zur Diagnose technischer Systeme.* Dissertation, University of Kaiserslautern (forthcoming)

[AMR90] Althoff K-D, Maurer F, Rehbold R. Multiple Knowledge Acquisition Strategies in MOLTKE. In: [WBG+90], pp 21-40

[AMW+89] Althoff K-D, Maurer F, Weß, S et al. Adaptive Learning in the Domain of Technical Diagnosis. In: *Proc. Workshop on Adaptive Learning*, FAW Ulm, FRG, 1989

[AMW+90] Althoff K-D, Maurer F, Weß, S et al. Die Lernkomponente der MOLTKE$_3$-Werkbank zur Diagnose technischer Systeme. In: [Mor91], pp 58-65

[Bar89] Bareiss, R. *Exemplar-Based Knowledge Acquisition.* Academic Press London, 1989

[BBP88] Bareiss R, Branting K, Porter B. The role of explanation in exemplar-based classification and learning. In: *Proc. Case-Based Reasoning AAAI-88*, 1988

[BW89] Breuker JA, Wielinga, BJ. Model-Driven Knowledge Acquisition: Interpretation Models. In: Guida P, Tasso, G (eds) *Topics in the Design of Expert Systems.* North Holland Amsterdam, 1989, pp 265-296

[Car83] Carbonell JG. Learning by analogy: formulating and generalizing plans from experience. In: Michalski RS Carbonell JG, Mitchell TM (eds) *Machine Learning*, vol 1, Tiogo Publishing Company Palo Alto, 1983

[Car86] Carbonell JG. Derivational Analogy in Problem Solving and Knowledge Acquisition. In: Michalski RS Carbonell JG, Mitchell TM (eds) *Machine Learning*, vol 2, Morgan Kaufmann, Los Altos, 1986

[Ham89] Hammond K. *Proc. of the 2nd DARPA Workshop on Case-Based Reasoning*. Morgan Kaufmann Palo Alto, 1989

[Kol83a] Kolodner JL. Maintaining Organization in a Dynamic Long-Term Memory. *Cognitive Science* 1983, 7: 243-280

[Kol83b] Kolodner JL. Reconstructive Memory: A Computer Model. *Cognitive Science* 1983, 7: 281-328

[Kol88] Kolodner JL (ed). *Proc. of a DARPA Workshop on Case-Based Reasoning*. Morgan Kaufmann Palo Alto, 1988

[Kot88] Koton P. Reasoning about evidence in causal explanations. In: *Proc. AAAI-88*, 1988, pp 256-261

[Mor87] Morik K. Sloppy Modeling. In: [Mor89a], pp. 107-134, 1987

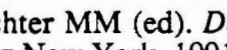
[Mor89a] Morik K (ed). *Knowledge Representation and Organization in Machine Learning*. Springer Berlin Heidelberg New York, 1989

[Mor89b] Morik K. Integration Issues in Knowledge Acquisition Systems. *SIGART Newsletter: Special Issue on Knowledge Acquisition* 1989, 108: 124-131

[Mor91] Morik K (ed). Sonderheft Maschinelles Lernen. *KI* 1991, 5: 1

[Off89] Offut D. SIZZLE: A case-based-reasoning system. Technical Report, Carnegie Mellon University, Pittsburgh, 1989

[Reh91] Rehbold R. *Integration modellbasierten Wissens in technische Diagnostik-Expertensysteme. Dissertation*, University of Kaiserslautern (forthcoming), 1991

[Ric91] Richter MM (ed). *Das*  *Buch*. Springer Berlin Heidelberg New York, 1991 (forthcoming)

[RKW89] Rissland EL, Kolodner JL, Waltz, D. Case-Based Reasoning. In: [Ham89], pp 1-13

[RW91] Richter MM, Weß S. Similarity, Uncertainty and Case-Based Reasoning in PATDEX. To appear in: *Festschrift for Woody Bledsoe*, Kluwer Academic Publishers, 1991

[RZ85] Rumelhart DE, Zipser D. Feature discovery by competitive learning. In: *Cognitive Science* 1985, 9: 75-112

[Sla91] Slade, S. Case-Based Reasoning: A Research Paradigm. *AI Magazine* Spring 1991

[Smi89] Smith LB. From global similarities to kinds of similarities: the construction of dimensions in development. In: [VO89], pp. 146-178

[SW88] Stanfill C & Waltz D. The memory based reasoning paradigm. In: [Kol88], pp. 414-424

[SZP90] van Someren MW, Zheng LL, Post W. Cases, Models or Compiled Knowledge: a Comparative Analysis and Proposed Integration. In: [WBG+90], pp 339-355

[Tve77] Tversky A. Features of Similarity. *Psychological Review* 1977, 84: 327-362

[VC89] Veloso M & Carbonell JG. Learning Analogies by Analogy - The Closed Loop of Memory Organization and Problem Solving. In: [Ham89]

[VO89] Vosniadou S & Ortony A. *Similarity and Analogical Reasoning*. Cambridge University Press, 1989

[WBG+90] Wielinga BJ, Boose J, Gaines B et al. Current Trends in Knowledge Acquisition. IOS Press Amsterdam, 1990

[Weß91] Weß S. PATDEX/ 2: ein System zum adaptiven, fallfokussierenden Lernen in technischen Diagnosesituationen. SEKI Working Paper SWP-91-01, University of Kaiserslautern, 1991

A Flexible Blackboard Architecture for Automatic Knowledge Elicitation form Text.

Pádraig Cunningham
Tony Veale

Hitachi Dublin Laboratory
Trinity College
College Green
Dublin 2
Ireland

Abstract

There is considerable interest at the moment on the use of knowledge based techniques for text analysis in large document databases. TWIG is an experimental system for automatic concept indexing of documents as they are scanned into a document filing system. Recently, TWIG has been restructured as a blackboard system in order to improve communication between the different reasoning processes in TWIG and to facilitate experimentation on the interaction of these processes. This paper describes this blackboard implementation and discusses the experiments that have been carried out with particular emphasis on the natural language processing components of the system. The blackboard implementation of TWIG uses an integrated knowledge representation formalism and the arguments in favour of this approach are presented.

1 Introduction

This paper describes the blackboard structure used in TWIG*, a system for knowledge elicitation form text. The research described here is part of the general trend towards using Natural Language Processing (NLP) and Knowledge Based techniques in information retrieval from large document databases. The objective of the research is for TWIG to produce a structured conceptual representation of the information in the documents as the documents are scanned into the system. The purpose of the derived

* twig twig. (coll.) v.t. and v.i. to observe: to understand.
[Poss. Ir. tuigim, discern: cf. Gael. tuig, understand.]

representation is to facilitate information retrieval and can be considered a conceptual index of the information in the text.

Recently the NLP and Knowledge Acquisition (KA) components of TWIG have been restructured using a blackboard shell called Ouija and the ramifications of this are the subject of this paper. A blackboard architecture is strongly indicated for the NLP and KA components in TWIG because of the different levels in the overall process and by the different reasoning mechanisms that are involved in reasoning between these levels. Our objective in adopting a blackboard structure is to produce a vehicle for experimenting with the organisation of the reasoning mechanisms. These mechanisms are introduced in more detail in section 3.

This objective of flexibility and experimentation has prompted us to adopt a homogeneous knowledge representation system for the NLP and KA components in the system. Essentially this has involved the integration of the lexicon that is the basis of the NLP system and the concept network that is the target of the knowledge acquisition process. The details of this integration are described in [3] so here we will restrict ourselves to presenting our justification of this somewhat contentious approach.

Section 4 contains an account of our perspective on blackboard organisation and an introduction to Ouija, our blackboard shell. Section 5 describes the implementation of TWIG in Ouija and the advantages accruing from that. First, the overall structure of the document filing system is outlined in the next section.

2 TWIG System Overview

The overall context of the TWIG knowledge acquisition system is best explained with reference to Figure 1. The objective for TWIG is to produce a basic conceptual representation of the information in documents for subsequent use in document retrieval. In the process of the image capture and analysis a rudimentary hypertext representation of the document is produced (partonomic links only). This hypertext is the basis of the concept indexing and the references produced in the concept indexing process will be to these hypertext nodes. So, in the context of Figure 1, TWIG is a knowledge based system that augments the concept network and fleshes out the hypertext by creating new reference links within the hypertext (see [2] for more details). Recently the knowledge acquisition system has been restructured within a blackboard system; this will be described in the next section.

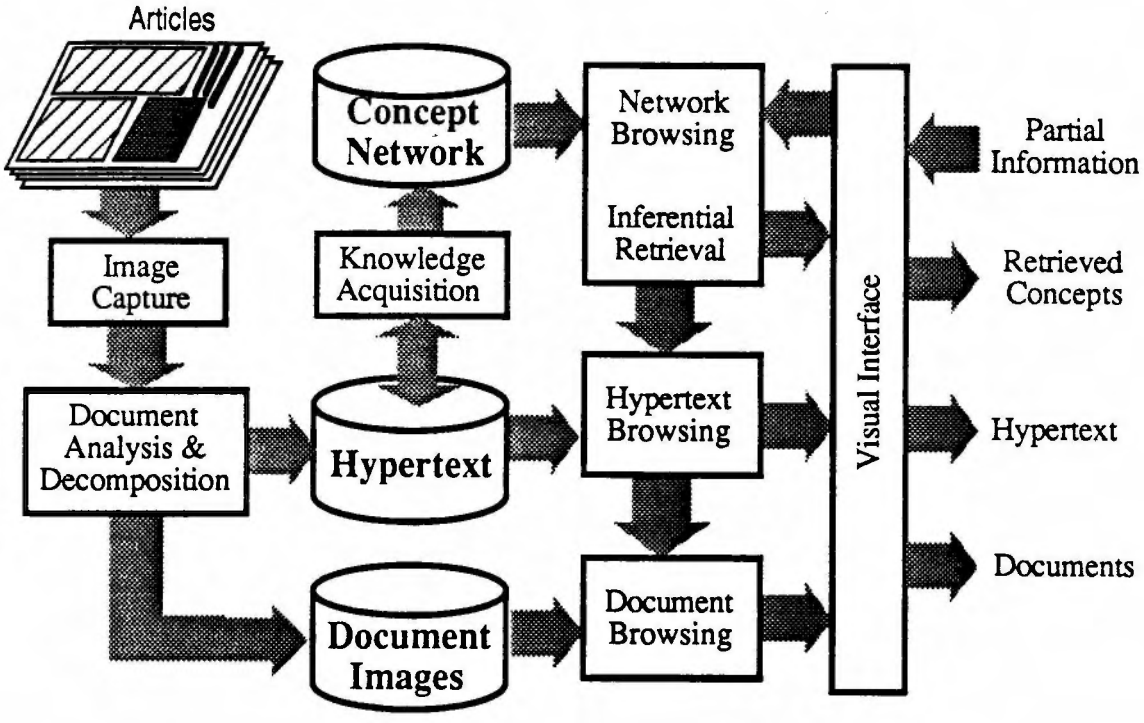


Figure 1. Overall structure of the document filing system.

3 A Blackboard Architecture for TWIG

Clancey argues that the terminology of blackboard knowledge based systems is a valuable perspective for understanding what every knowledge based system is doing. Further, the blackboard architecture is a useful formalism for structuring a complex knowledge acquisition task and facilitates the developer in formalising the part of the knowledge base that can subsequently be reused [1]. In particular the blackboard architecture is appropriate when the reasoning process is complex, where the knowledge is processed at different levels of granularity within the system and when the reasoning can be partitioned into different knowledge processes. Knowledge elicitation from text conforms to these criteria; it is essentially a natural language processing problem that has various components. These components reflect the interacting strata of language and cannot be performed sequentially. This argument is presented in [4] by Sowa who suggests that these components can be implemented as coroutines that allow feedback between different processing stages. The blackboard architecture used in TWIG addresses this requirement for communication between the processes quite well.

Figure 2 illustrates the interaction between the different process in TWIG communicating using a blackboard structure. Brief descriptions of the various processes

are as follows:-

- **Phrase Preprocessor:** The fact that TWIG uses a Unification Grammar allows for some preprocessing that uses unification to group words into phrases. This chunking speeds up the subsequent parsing.

- **Parsing:** The initial NL processing in TWIG is performed using a chart parser operating with a unification grammar. The output from the parser is a structured semantic representation of the text.

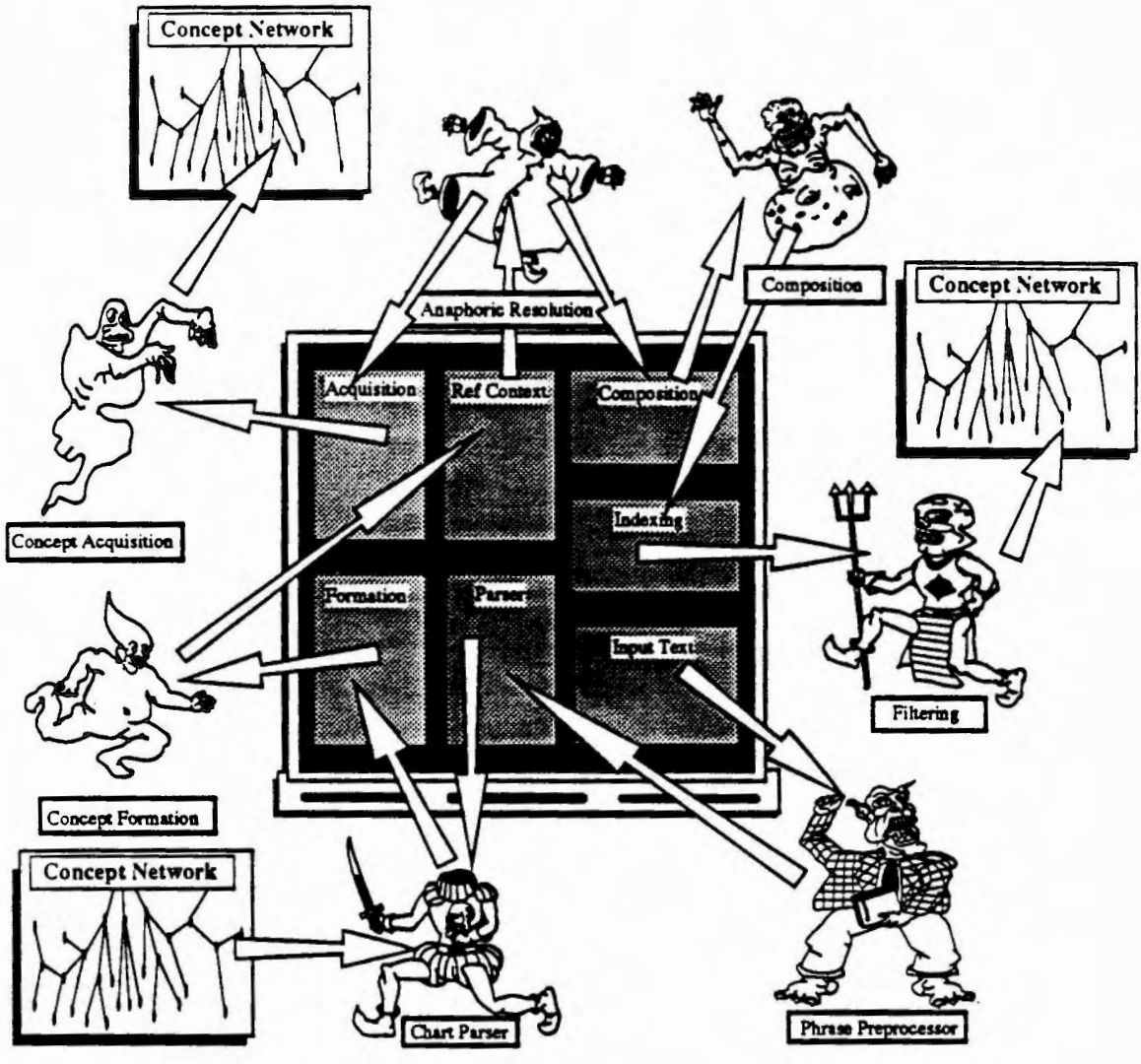


Figure 2. A blackboard architecture for knowledge elicitation form text.

- **Concept Formation:** Conceptual structures are derived from the semantic structures produced by the parser. These conceptual structures are frame representations of the corresponding semantic structures.

- **Concept Acquisition:** Adds new knowledge that has been discovered in the knowledge acquisition process to the concept network.
- **Anaphoric Resolution:** resolves references in the text; works on the conceptual structure.
- **Composition:** Composes the product of the concept formation into more complex structures; creates links between related substructures.
- **Filtering:** Determines which additions will actually be made to the concept network.

3.1 Implications for Knowledge Representation

It is worthwhile implementing a knowledge based system within a blackboard architecture if separate reasoning mechanisms can be identified in the reasoning process, and if the requirement for communication between these processes is complex. Further, if this blackboard system is to be flexible then it is desirable to adopt a common formalism in which the different agents can communicate on the blackboard. In TWIG there are two major knowledge categories: the encyclopaedic knowledge of the concept network and the semantic knowledge of the lexicon, which is used in parsing the text. This knowledge has been integrated into one representation so that, effectively, the concept network and lexicon are merged to produce one integrated representation. In effect, different knowledge processes drawing on this representation are working at the same level of abstraction but at different levels of granularity. The lexicon and concept network contain similar overlapping information and the integration allows this to be maintained centrally.

It is worth locating this approach to knowledge representation in the general context of research on knowledge acquisition from text. Figure 3 illustrates a conventional interpretation of the stages in text understanding. The Conceptual Structure is the target machine representation of the information in the text. There are two intermediate representations; the Surface Structure and the Deep Structure. Our integration of the Lexicon and Concept Network mean that the Deep Structure and the Conceptual Structure are represented using the same formalism. So, in the context of Figure 3, our parser produces output that is somewhere between the Deep Structure and the Conceptual Structure in level of abstraction. There is a difference between the output from the parser and the final representation but the difference is one of granularity rather than level of abstraction. This is illustrated in Figure 4 which shows that, in Twig, processing subsequent to parsing works within a single representation formalism.

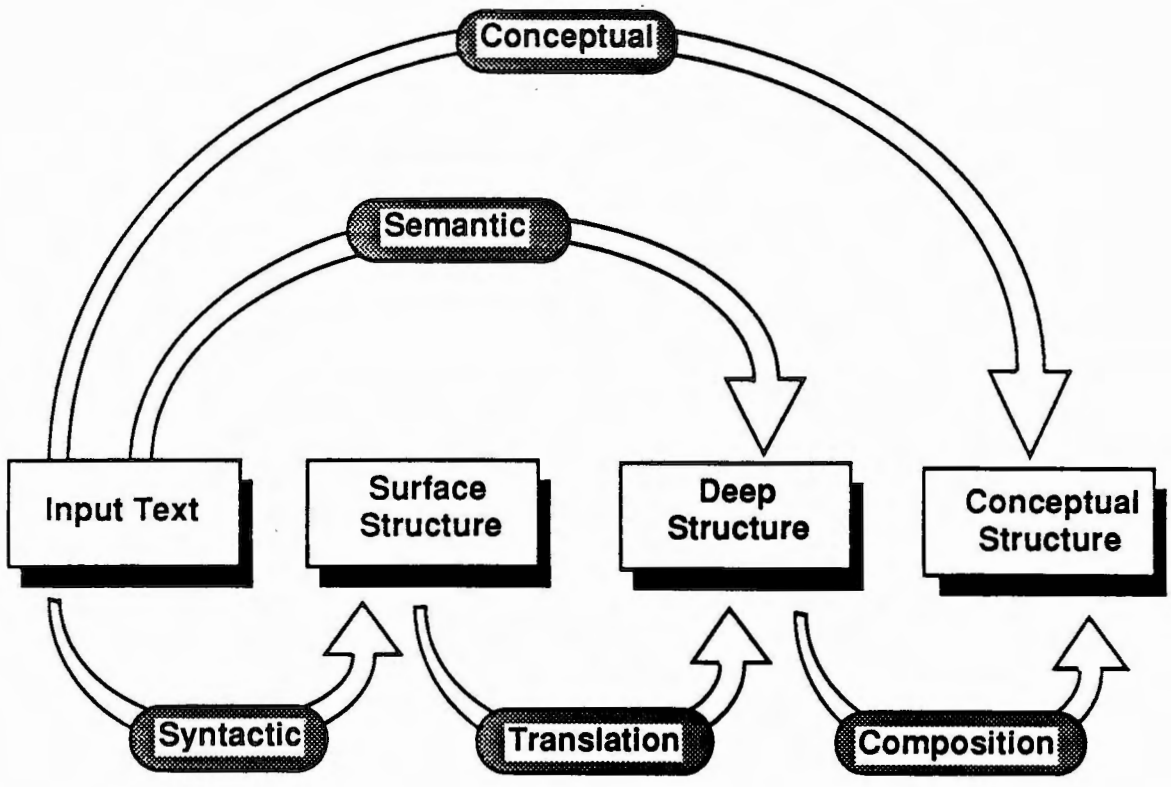


Figure 3. A conventional overview of knowledge acquisition from text.

On the face of it this is similar to the approach in Schank's work [8] [10] but there are some important differences. Schank's approach to parsing relegates syntax to a secondary rôle whereas our parser is a chart parser working with a unification grammar. Indeed, in section 5 there is a brief account of the effect of dropping this syntactic analysis from the reasoning process.

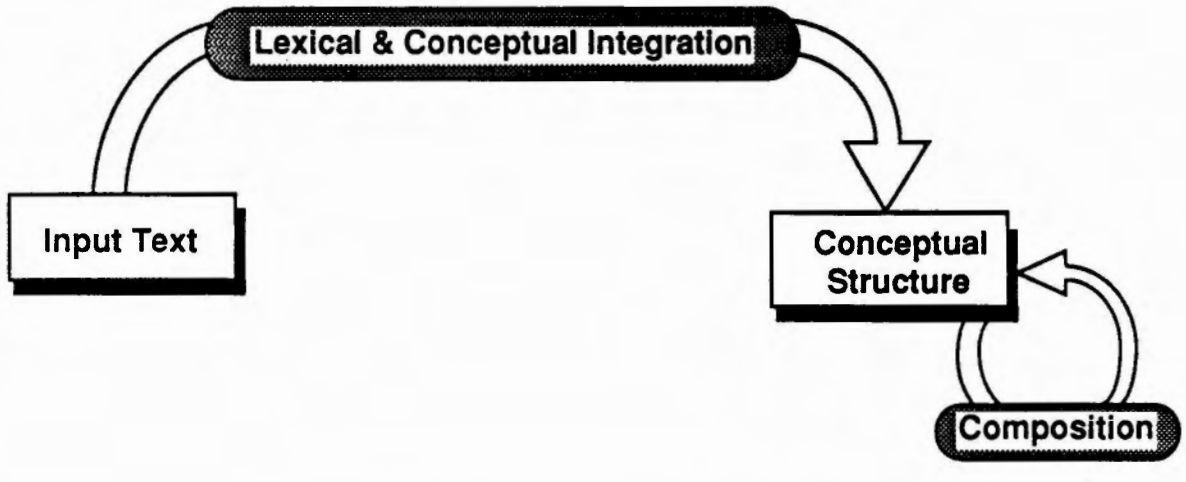


Figure 4. Knowledge acquisition from text as done in TWIG.

Finally, the "Conceptual" arrow in Figure 3 can also represent text skimming. Skimming is a template driven approach whereby text is scanned seeking relevant information to fill the slots in the template [9]. The core knowledge base in TWIG can act as a structured template for text skimming and some experiments in TWIG on this issue are discussed in section 5.

Before continuing with the description of the blackboard implementation of our knowledge acquisition shell we will take a step back and consider blackboard structures in general.

4 Blackboard Control Architectures

The Blackboard model is a control metaphor employed for the distribution of problem solving effort among a variety of disjoint (and possibly competitive) knowledge sources. As such, a blackboard consists of a common data structure through which these knowledge sources communicate, and an associated control mechanism through which that communication is moderated. It is the form and function of this moderator which varies most from one blackboard implementation to another. Moderator architecture can be classified broadly as fully distributive demon-driven control, meta-level hierarchical control and explicit dedicated control architectures [7].

Demon-Driven Control

Such moderator architectures are said to be event-based. An action such as writing data onto a blackboard panel constitutes an event, which may activate the appropriate knowledge sources associated with that panel, implemented as demons sensitive to such events. These demons in turn trigger other events, causing other demons to become activated, and so on, to the extent that blackboard control is distributed among the demons themselves. Should several demons be activated by the same event, scheduling may become an issue.

Meta-Level Hierarchical Control

Blackboards employing such control architectures organize knowledge sources as a hierarchy of meta-level (control oriented) and domain (task oriented) actions. Leaf nodes in the hierarchy represent domain actions while interior nodes represent meta-level control decisions, governing the evaluation of subordinate nodes.

Explicit Dedicated Control

Explicit control architectures employ dedicated knowledge sources as schedulers to govern the application of related knowledge sources. Blackboards employing such dedicated control architectures achieve greater flexibility as the control mechanism is distributed among various knowledge sources, which can organize themselves dynamically to suit the task situation.

Note that these categorisations are necessarily broad to encompass the spectrum of different blackboard architectures; a given implementation may resist classification in full under just one of these headings, employing instead a mix of ideologies to suit the task at hand.

It is this problem of demarcation we seek to redress with the Ouija board architecture. We define a Ouija board as a generalized blackboard architecture employing demon-based event-driven control, but which offers no policies on demon scheduling or division of labour among knowledge sources (task vs control). Events such as panel I/O trigger the corresponding KS demons, the execution order of which is arbitrary and without formal specification. Thus, the Ouija board * is a common denominator blackboard system upon which more exotic moderator/control mechanisms may be built.

It is easy to see how both meta-level control and explicit dedicated control architectures can be implemented using Ouija. The blackboard designer simply allocates dedicated control demons to coordinate the actions of other task-oriented demons, via specially allocated blackboard panels which act as *polling stations* through which the *minion* processes voice their views. In turn, the *overseer* demons may respond to the control of others, forming a hierarchy of disjoint knowledge-based moderators.

4.1 Ouija-Board Organization

The following is a brief outline of the Ouija-board shell as implemented in the TWIG system. As defined earlier, a Ouija-board is a demon-driven blackboard, which in turn consists of a collection of write-sensitive I/O panels. A write operation onto a blackboard panel activates the associated demons which *haunt* that panel; in effect, demons intercept the actual data before it is written and thus have the power to authorize, deny or generally modify information submitted to their host panels.

A demon is defined as an autonomous process which monitors communication to

---

* The fundamental principle of the Ouija architecture is that of control via autonomous knowledge agents, with no rigid moderator policy enforced by the system. Designers must provide their own control mechanisms and scheduling policies within the demon framework of the architecture. It is from this somewhat arbitrary view of demon organization that the system is named.

one or more blackboard panels. Therefore, to specify the form and nature of Ouija demons, it is first necessary to specify the structure and associated access functions of a blackboard panel. Because a blackboard serves as a common reservoir of heterogeneous data among different knowledge sources, panels must support a variety of storage forms and access methods. For instance, one panel may be employed as a stack, queue or generic list structure, while others may find service as hash tables, arrays or frames. Indeed, the same panel may have to provide a variety of storage perspectives to different demons acting upon it. Thus, to achieve maximum flexibility, Ouija panels are implemented as direct lookup tables with individually accessible cells; individual cells may contain any LISP object and are addressed by panel name and key value.

This enables us to refine the trigger component of a demon to such a level that it becomes write-sensitive to a specific set of keys only. In turn, this enables different demons to perceive different structural views of the same panel and react to different hotspots within that panel.

Blackboard Storage/Retrieval Services

BB-Create-Panel (panel-name size)
Create new blackboard panel.

BB-Clear-Panel (panel-name)
Clear panel of blackboard.

BB-Map-Panel(virtual-panel, actual-panel)
Transparently re-route all I/O from virtual-panel to actual-panel.

BB-write (panel-name key data)
Write data into an indexed cell of a blackboard panel.

BB-push (panel-name key data)
Push data onto an indexed cell (LIFO) of a blackboard panel.

BB-read (panel-name key)
Return value stored in a blackboard panel cell specified by key.

BB-read-first (panel-name key)
Return first value stored in a blackboard cell (LIFO) specified by key.

BB-read-all (panel-name)

Return a list containing all cells in a particular blackboard panel.

BB-read-selected (panel-name key-list)

Return a list containing selected panel cells matching key-list.

4.2 Demon Specification

Demons can be defined in a hierarchical* fashion, allowing the system designer to create abstraction demons from which others will inherit common functionality. For instance, a demon class Reporter-Demon can be created which simply sends notification of its activation (with its relevant parameters) to a particular window stream whenever a subordinate demon is triggered. Thus, while such subordinates will have their own local function, they also inherit the functionality of the Reporter-Demon; disabling the Reporter-Demon will in turn disable automatic notification from subclass demons.

Hierarchical definition of demons greatly improves the system modularity offered by the use of a blackboard. This modularity allows the designer to experiment with different components (such as a new parser module - for instance, a WASP parser instead of a chart parser) without effecting other modules. Provided the new module presents the same interface to the blackboard, no other demons will need to be changed. Such independence is not offered by the traditional *invoke-by-direct-procedure-call* nature of conventional implementation.

Demon Control Services

Create-demon (demon-name <Body>)

Create a particular demon (class or instance)

<Body> contains specifications of demon in frame (slot:filler) format.

Sleep (demon-name)

Disable a particular demon - sleeping demons can not be triggered.

---

* The Ouija shell has itself been implemented using the TWIG concept network. As such, panels and demons are first class objects (concept frames) in the system, and may be browsed using existing TWIG services.

Wake (demon-name &key catchup)

Re-enable a particular demon. If the :catchup mode is specified, the demon is activated for every entry written to the blackboard while the demon has been sleeping - this is an extremely useful way of delaying computation until sufficient data is available.

Demon Activation

Whenever a demon is activated, its overall function (local and inherited) is performed, the code of which has access to the following special variables which are bound within the environment of the demon:-

***Demon**	- name of the triggered demon
***Panel**	- name of the *hot* panel which triggered the demon
***key**	- the key used in accessing the *hot* panel
***data**	- the data written to the *hot* panel
***stream**	- I/O window associated with *hot* panel - especially useful to reporter demons.

5 Experimenting with Ouija

Having outlined the essential blackboard architecture adopted in Ouija, we now wish to demonstrate its flexibility as a platform upon which the various TWIG processes are coordinated. As previously described, the Ouija formalism enables inter-process communication among knowledge agents which are essentially *cooperative strangers*. This interdependence affords great modularity in the distribution of effort among demons.

For instance, suppose one was to remove the phrase preprocessor demon from the TWIG scheme of things; after all, such a component is heuristic rather than grammar driven, essentially making a trade-off between speed and accuracy. Several options present themselves: primarily, we could rewrite the preprocessor demon to do no work at all, and simply off-load its input directly into the parser panel of the blackboard. However, to prevent useful preprocessing code going to waste, we could simply maintain a flag indicating whether the preprocessor demon should process its input or effect a direct transfer to the parser panel. More useful, however, is a solution employing two mutually

exclusive demons, Preprocess and Offload, which are never active at the same time. Wake(Preprocess) & Sleep(Offload) prompts TWIG to perform initial phrase prepackaging upon all input text, while Wake(Offload) & Sleep(Preprocess) places the full burden of parsing squarely in the domain of the parser demon.

The benefits of demon modularity are more immediately obvious, however, in the choice of parser employed by the system. Presently, TWIG incorporates a chart parser operating within a unification grammar, but such a configuration is not at all dictated by the KA task. Experimentation with other parser models and grammar formalisms can be performed with ease within the blackboard framework; we need simply plug out one parser and plug in a replacement, such as a WASP or an ATN. As all interaction with the parser module is performed via a specified blackboard panel, communication with other components does not suffer. Indeed, the designer may conclude that a parser component is not warranted at all, for the use of a phrase preprocessor in conjunction with the composition component forms the makings of a good semantic *skimmer* in itself. Again, several options present themselves to achieve such a configuration: we could reorganize the blackboard and eliminate the parser panel altogether, or we could redefine the parser module to off-load its input directly into the composition panel of the blackboard. The most elegant solution, however, simply maps the parser panel onto the composition panel, using Map-Panel(Parser, Composition), with reference to Figure 2. Such a mapping is totally transparent to all demons involved, where all I/O to and from the parser panel is automatically rerouted to the composition panel. An experiment with such a skimmer in TWIG supports the deactivation of the syntax component as a considerable source of improvement in efficiency, at the cost of accuracy in the overall understanding process. In particular, the disconnection of the syntax module caused much deterioration in the system's ability to disambiguate multiple word senses.

Effectively, panel mapping adds an extra layer of abstraction to the blackboard model (in particular, Ouija), enabling the creation of *virtual* panels; this in turn reduces the independence between panels and demons at the cost of additional dependence among panels.

6 Conclusions

The blackboard approach to knowledge based system development is useful both as a perspective for the designer of the KBS and as an explicit control structure for the KBS. The Ouija blackboard shell emphasises the autonomy of cooperating knowledge sources by delegating distributive control among demons.

An argument central to this paper is that if many knowledge sources are operating with the same representation formalism there is considerable scope for experimentation

with the interaction of the components of the system.

Currently we are using this flexibility to explore the interaction of the different processes in the TWIG system; for instance the merits of skimming v's parsing in the NLP process.

References

1. Clancey W. J., 'Implications of the system-model-operator metaphor for knowledge acquisition', in Proceedings of the First Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop, ed. H. Motoda, R. Mizoguchi, J. Boose, B. Gaines, pp65-80, 1990.

2. Cunningham P., Fujisawa H., Hederman L., Cummins F., 'A combined approach to text retrieval using concept networks linked to hypertext', in Proceedings of the First Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop, ed. H. Motoda, R. Mizoguchi, J. Boose, B. Gaines. pp235-248, 1990.

3. Cunningham P., Veale T., 'Organizational issues arising from the integration of the lexicon and concept network in a text understanding system', submitted to IJCAI'91, 1991.

4. Fujisawa H., et al., "Document analysis and decomposition method for multimedia contents retrieval", Proceedings of the Second International Symposium on Interoperable Information Systems (INTAP), pp231-238, 1988.

5. Sowa J.F., Conceptual Structures, Addison Wesley, 1984.

6. Thagard P., Computational Philosophy of Science, MIT Press, Cambridge London, 1988.

7. Kuru S., Bek F., "Goal-driven blackboard control architecture based on extending partially complete general goal trees", Knowledge-Based Systems, pp236-241, Vol. 3, No. 4, 1990.

8. Schank R., , Conceptual Information Processing, North Holland, 1975.

9. Jacobs P.S., "To parse or not to parse: Relation-driven text skimming", COLING-90, pp194-198, 1990.

10. Selfridge M., "Integrated processing produces robust understanding", Computational Linguistics, Vol. 12, No. 2, 1986.

Architectural Issues in KA Tools: Towards Structured Transformation into Knowledge-Bases

Henrik Eriksson

Department of Computer and Information Science
Linköping University, S-581 83 Linköping, Sweden
E-mail: her@ida.liu.se

Abstract

Tool-supported knowledge acquisition can typically be seen as a transformation process from knowledge structures elicited from domain experts into knowledge bases. This process might involve several intermediate steps. The specific transformation strategy employed in early stages differs among various classes of knowledge acquisition tools due to variation in the knowledge acquisition method applied. However, in the later stages of the process common architectural approaches to *knowledge base generation* in knowledge acquisition tools may be considered.

This paper brings such architectural aspects of the transformation process into focus. A systematic approach to the generation of knowledge bases from knowledge acquisition systems is presented. The basic idea is to separate the generation mechanisms from the specification of what knowledge base structures should be generated. A generation mechanism guided by transformation rules is used to transform knowledge structures elicited into the target format for delivery.

1 Introduction

Many efforts in tool support for knowledge acquisition (KA) have been reported during recent years. However, surprisingly little has been published on specific architectural components common to many classes of KA tools.[1] Clearly, evaluating and comparing various approaches to KA tool support is a serious problem. Nevertheless, evaluations and comparisons of subsystems can be made and, hence, provide guidance for subsequent generations of KA tools.

This paper focuses on one important task for a KA tool, namely the transformation process from knowledge structures elicited[2] (e.g. "near-expert" representations or results from machine learning) into target knowledge bases for delivery. In the approach taken, the target knowledge base is generated through a set of *transformation rules*, which map source structures into the knowledge base format required. In other words, the generation mechanism is *parameterized* by transformation rules. This type of transformation system provides (1) a flexible representation of the mapping into target structures; (2) improved maintainability both for the generator and the knowledge acquisition tool as a whole, since the interpretation of elicited structures can easily be changed at a later time; (3) a possibility to add domain knowledge to the transformation process; (4) support for multiple target formats; and (5) transformation of knowledge structures elicited to conventional programming languages for efficient run-time versions.

[1] A notable exception is a description of the *Dialogue Manager* subsystem of AQUINAS provided by Kitto and Boose [13].

[2] The terms *knowledge acquisition* and *knowledge elicitation* are treated as synonyms in this paper.

Compared to straightforward implementation in procedural code, the separation into a generator mechanism and a transformation-rule base that constitutes the mapping from structures elicited to knowledge bases has many advantages, most notably flexibility and maintainability. There are also additional requirements on transformation systems. In particular, support for *multiple target formats* (i.e. different languages or shells) and smooth *regeneration* of knowledge bases for incremental knowledge acquisition and development. Another important issue in this context is integration of generated output with manually encoded knowledge bases and specialized inference mechanisms.

This paper is structured as follows: Section 2 provides a background and stresses the importance of a systematic approach to knowledge base generation as well as the requirements on such an architectural component. Section 3 distinguishes generated and manually encoded parts of the knowledge base to define the scope of knowledge base generation; Section 4 introduces the idea of generation through transformation rules; Section 5 provides a framework for multiple target formats; Section 6 outlines a generator for incremental regeneration of the target knowledge base; and Section 7 discusses integration issues for target knowledge bases. Finally, the knowledge base generation approach is summarized and discussed in Section 8.

2 Background

Although internal transformations and knowledge base generation from KA tools might seem remotely related to the actual issue of knowledge acquisition from domain experts, such questions are—as we shall see—important enabling technology for providing productive KA tool support.

2.1 Motivation

An important task of a knowledge acquisition tool is to transform knowledge structures from a representation close to what can be elicited from experts (by some appropriate method or technique) into the required output format for the tool. In many cases the output required is a knowledge base with some structure and syntax, but the output can also be knowledge in another format for consideration by experts. In many experimental KA tools focusing on different aspects of elicitation, machine learning, etc.; the manner in which the transformation and knowledge base generation task is performed is of less importance for the research goals in question.

However, as tools are being specialized for various domains and used on a production basis, this architectural issue is becoming increasingly important. Manual implementation of knowledge acquisition tools can benefit significantly from a systematic strategy for knowledge base generation. For generation of knowledge acquisition tools from *meta-tools* [20; 8; 9] a systematic architecture for the knowledge base generator is required. To avoid dependency on the target problem-solving mechanism for KA tools generated, it is important to provide an architecture where the generation mechanisms are separated from the specification of what knowledge base structures should be generated from knowledge elicited by a target KA tool.

2.2 Requirements

A systematic approach to internal transformations and the knowledge base generator is highly desirable in manual development and required in automatic generation of KA tools. Preferably, transformation mechanisms should offer a high degree of maintainability (which would imply some form of declarative specification of the transformations).

Knowledge engineers might want to change the target knowledge base format for KA tools developed, i.e. *retarget* KA tools (see Section 5 for a discussion of techniques for multiple target formats). An example of such a situation is when separate development and production environments are used for knowledge bases generated. For this purpose it is desirable to reuse as much as possible of the transformation-rule base. If an intermediate representation for knowledge bases is introduced, target-dependent parts of the transformation-rule base can be separated from target-*in*dependent parts (see Section 5).

A practical problem in gaining an executable system from knowledge acquired is the amount of time it takes to generate and load knowledge bases. Adding support for *incremental* generation (i.e. generate only changes made since the last generation) can reduce the generation and load time for large knowledge bases (see Section 6). Thus, experts can be given rapid feedback in terms of running prototype versions of the system.

2.3 Method

The approach to transformation and knowledge base generation presented in this paper is the result from several design iterations and experimentations in this area. This particular architectural component has been implemented, evaluated, revised and refined in several iterations.

An initial version (which itself required several design iterations) of the generator component described was first used in the knowledge acquisition tool P10 [9]. As part of a subsequent project (the meta-tool DOTS) the transformation system and the generator was completely rewritten to function as an integrated component in target KA tools. DOTS [8; 9] uses the same basic generation mechanism to produce both target KA tools from specifications entered by knowledge engineers and knowledge bases from knowledge acquired by target KA tools. The principal difference is that the generator is parameterized with different sets of transformation rules. Compared to P10, the generator in DOTS has facilities for multiple target formats (Section 5) and incremental regeneration of knowledge bases (Section 6).

3 Generation model

When discussing the transformation from KA tools into knowledge bases it is important to keep in mind that what is wanted is *performance* at some level, not necessarily knowledge bases (see Section 4.4 for a discussion of alternative approaches). There are other alternatives to actually generating knowledge bases, for instance, interpreting structures elicited directly in the KA tool.[3] Many hybrid tools and shells offers a "fast-load" format for efficient save and load of knowledge bases. It is, in principle, possible to generate such representations too. However, such fast-load formats are subject to change between releases of hybrid tools and shells. Apart from knowledge bases (or equivalent), other types of output from KA tools can also be required, e.g. automatically generated (paper) documentation for knowledge acquired.

Since this paper is primarily concerned with transformation into target knowledge bases, i.e. knowledge base generation, the overall generation model as well as alternative approaches will be discussed. There are various approaches to producing output from KA tools and save elicited structures persistently on file, which is required in practicable KA tools. Figure 1 depicts three possible strategies: (a) the knowledge base generated (which can be stored persistently) can be read back into the KA tool, i.e. there exists is a *reverse* transformation mechanism and the knowledge base can be "edited" by the

[3]The KA tool KAVE comprises a *simulator* that allows interpretation of rules (in the KA tool) without actually generating knowledge bases [24].

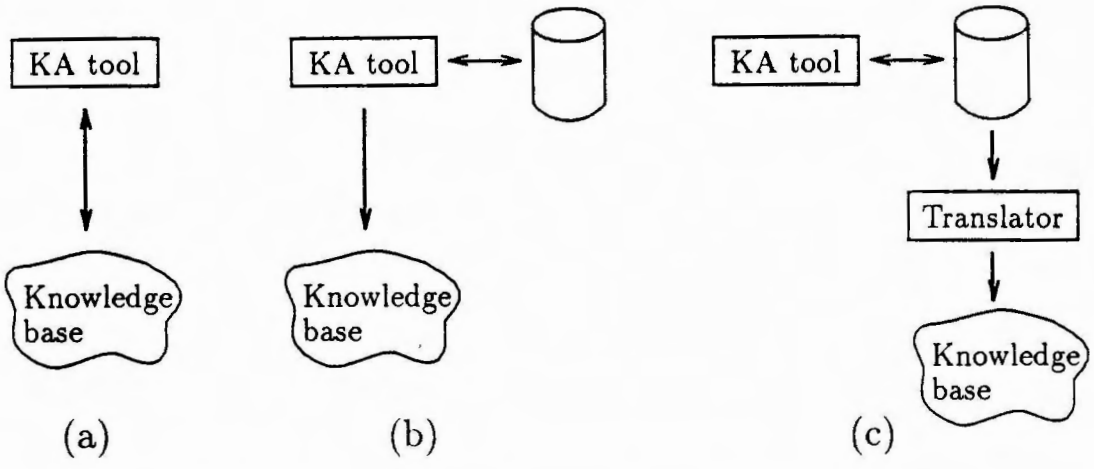


Figure 1: *Different ways of generating knowledge bases and store knowledge acquired persistently on file for later refinement (and regeneration of the knowledge base). A bidirectional transformation between the internal representations in the KA tool and the knowledge base can be used to avoid separate mechanisms for persistent storage (a). By using a separate for persistent storage on file, a unidirectional transformation becomes sufficient (b). The generator can also be separated from the KA tool, i.e. by using a translator that generates a knowledge base from what is stored on file (c).*

KA tool; (b) the transformation into knowledge bases is unidirectional and the internal knowledge representations (at some level) in the KA tool can be stored on persistent files directly and be read back again; and (c) structures elicited are first stored persistently in file and this file is transformed into a target knowledge base by an independent translator.

The advantage with approach (a) is that there is no need to implement[4] persistent storage for the internal (intermediate) representation in the KA tool. However, a major drawback is that a mechanism for the reverse transformation is required. Even though such mechanisms can be implemented there is still principal problems since information may be lost in the transformation process, e.g. if there are many-to-one mappings between what is elicited and components in the knowledge base there is no reverse transformation. KAVE [24] is an example of a KA tool that implements strategy (a). KAVE can be seen as a graphical rule editor, since there is a fairly direct mapping between what is entered in the forms and the rules in the knowledge base.

Approach (b) involves persistent storage of some intermediate representation in the KA tool. The obvious disadvantage is the effort required to implement storage mechanisms (if they are not provided by the software infrastructure). The major advantage is that the reverse transformation can be avoided. P10, DOTS and several KA tools developed by others implement strategy (b). In approach (c) the transformation mechanism is broken out in a separate translator (see Figure 1c). This means that the translator must be able to read what is stored on file from the KA tool. If the format for persistent storage can be standardized, an independent (generic) translator common to many KA tools can be used.

Generally, knowledge acquisition tools can not be expected to support all aspects of a domain. It is more reasonable to separate the target knowledge base into *generated*

[4] A process which can be painful if the internal representation contains circular structures or if it is only meaningful to store parts of the structure.

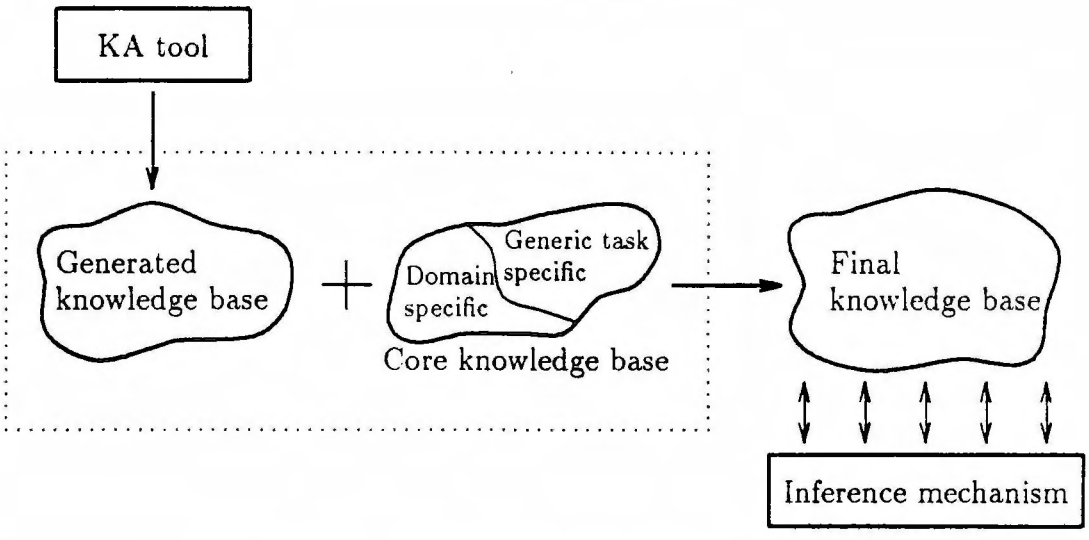


Figure 2: *Generated and core knowledge bases.*

and *core* (or hand-crafted) parts (see Figure 2). Core knowledge bases can also be seen as library knowledge bases for generic tasks and domains covered by a KA tool. Thus, core knowledge bases should be generic with respect to knowledge bases generated. From the perspective of KA tools, core knowledge bases in combination with inference engines can be seen as the target environment for knowledge bases generated. Core knowledge bases can even be more or less unique if domain-oriented knowledge acquisition tools are applied.[5]

In environments that allow a mixture of generated and hand-coded parts, there is naturally a trade-off between manual knowledge engineering and extension of KA tools to handle unique cases in new domains. It is usually better to acquire and code rare knowledge structures and unusual cases manually, while the tool should emphasize support for repetitive patterns of knowledge.

Core knowledge bases can be further divided into *generic-task-* and *domain*-specific parts. In the generic-task-specific part, knowledge common to the generic task [5] supported is stored, e.g. general planning or troubleshooting knowledge. For knowledge acquisition tools supporting particular generic tasks, this part of the core knowledge bases should have the same scope as the KA tools. Generic-task-specific knowledge bases should also be generic with respect to tool-elicited knowledge and knowledge base structures generated from KA tools.

In the domain-specific part of the core knowledge base, exceptional cases for each domain can be represented. This part is unique for each application and does not have to be organized to meet any generic requirements. However, it must be compatible with the generic-task-specific parts and parts generated from the KA tool.

The division of core knowledge bases into generic-task- and domain-specific parts should not be taken strictly. For instance, in domain-specific KA tools core knowledge bases can be tailored to the reasoning strategy for each particular domain, i.e. there is no clear borderline between generic-task- and domain-specific components. In this case core knowledge bases can be augmented with application-specific knowledge manually.

[5] Note that core knowledge bases correspond to *common libraries* in the context of KA tools generation from meta-tools [9].

As depicted in Figure 2, the final knowledge base is interpreted by an inference mechanism, which may be specific to the generic task in question. The inference engine (possibly together with the generic-task-specific part of the core knowledge base) is often referred to as the *performance system* for the KA tool. MOLE$_p$ is the performance system in MOLE [10] and YAKA$_p$ is the performance system in YAKA [19, Section 8.2.2]. An empty version of ONCOCIN, e-ONCOCIN, is the target environment (and the performance element) for PROTÉGÉ-generated KA tools [20].

4 A transformational approach

Transformation rules can be used to map knowledge structures from the tool into entities in the target knowledge base. Generally, transformation rules can be seen as a (partial) mapping from program to program [21; 11]. Transformation rules can either be represented as an algorithm (procedural rule) or as schematic (or pattern replacement) rules. For knowledge base generation purposes we suggest a special form of transformation rules. The premises of such transformation rules are adapted to knowledge representations in knowledge acquisition tools, whereas the conclusions are analogous to templates for target knowledge-base structures. Transformation rules have successfully been used in the P10 and DOTS projects to define the mapping between elicited structures and target knowledge bases, i.e. to parameterize the knowledge base generators.

4.1 Transformation rule format

The premise of the transformation rule format suggested is intended to *describe a situation* in the internal representation in the KA tool where certain structures should be added to the target knowledge base (i.e. when a transformation should be applied). Therefore, it should be possible to specify such situations in a representation conveniently, i.e. in the usual representation language (e.g. Lisp). The conclusion of each transformation rule can be a *template* which is filled in with appropriate values at generation time. Such values can originate from knowledge elicited, i.e. from the internal representation. Preferably, templates should resemble the syntax of the target knowledge base format (which can improve readability). The generality of transformation rules varies. If internal representations in KA tools are structured hierarchically (i.e. object-oriented), the *scope* of each transformation rule can be defined with respect to classes in the hierarchy. Thus, the basic format of a transformational rule may appear as follows:

```
(defrule name
         scope
         premise
         (transform symbol "target knowledge base code"))
```

where *name* is an arbitrary name of the rule, *scope* is a list of quantifiers in the format (*var* `:in` *list*), *premise* refers to the tool structure (e.g. a Lisp-expression), *symbol* is the parameter to be asserted, and *target knowledge base code* is the concluded value. If the *premise* evaluates to non-nil, *symbol* will be asserted to the *target knowledge base code* string. The knowledge base generator will collect the conclusions of all instances of the rule where *premise* is non-nil on *symbol*.

The conclusion string (*"target knowledge base code"*) deserves further explanation. It is possible to include parameter names in the string, which are replaced by the value of the corresponding parameter at the time the conclusion is being drawn, i.e. the string works as a template. Consider the following example:

Say that the parameter `a` has the value `x`, and that the parameter `b` has the value `(i j k)`.

If a transformation rule makes the following conclusion:

```
(transform target-rule "if {a} and ({b* or}) then...")
```

then run-time variable substitution would assert the value of `target-rule` to the following:

```
"if x and (i or j or k) then..."
```

In the implementations of a transformation rule mechanism we have used arbitrary Lisp-expressions are allowed in the premise and Lisp-expressions plus special forms (e.g. `transform`) are allowed in the conclusion.

Different knowledge acquisition tools would need different transformation and knowledge base generation strategies. Transformation rules are a fairly general approach since the input representation may vary. As illustrated in the next section, the starting point for the transformation can be a matrix representation—a common output of several grid techniques. The starting point may also be a frame-based representation where values are stored in slots. This may, for instance, be the output from knowledge acquisition through graphical forms. A drawback of transformation rules is that they are unsuited to tasks involving *parsing* of structures, i.e. analyzing the surface structure of a (formal) language.

4.2 The generation process: examples

This section exemplifies the transformation process into rules by providing two examples. In the first example the starting point is a matrix, which can be a decision table edited directly by experts or the result of elicitation according to some grid-oriented technique. The second example involves transformation from a domain-specific form (filled in by an expert) into the resulting rule in the knowledge base.

4.2.1 Starting with a matrix representation—decision tables

This example illustrates the generation process by first describing the source, in this case a decision table filled in by the expert. Secondly, the application of transformation rules for knowledge base generation of this decision table is outlined. Finally, the resulting structure—in this case a rule—is presented. (This example is a simplified version of an actual case that occurred the P10 project.)

A sample decision table filled in by the expert is shown in Figure 3. (Note that similar matrices are the output of other knowledge acquisition techniques as well.) The current contents of this table—which is taken from a biochemical domain—states that if the number of SDS bands is ≥ 2 and the protein purity is unsatisfactory, the current strategy should be to prefer node 3 (in an action graph). After the expert has filled in the decision table, the contents are stored in an internal representation, or *knowledge module* (see Section 6.2). In P10 and DOTS such knowledge modules are used as a starting point for the transformation into knowledge bases.

A simplified transformation rule is shown in Figure 4. This sample rule has the name `decision-table-1` and is (in a more complete form) used in the P10 system to generate rules from a decision table filled in by the expert. The scope of the rule ranges over the columns (A–D) in the decision table (cf. Figure 3). The premise of the transformation rule is used to check if there is (a) an item in the column in question of the table (`premise`) and (b) a suitable conclusion for that column (`strategy-seq`). If the premise is true, i.e. a column and a conclusion is filled in, a corresponding target rule is generated according to the conclusion. The conclusion of `decision-table-1` is a rule schema for a hybrid

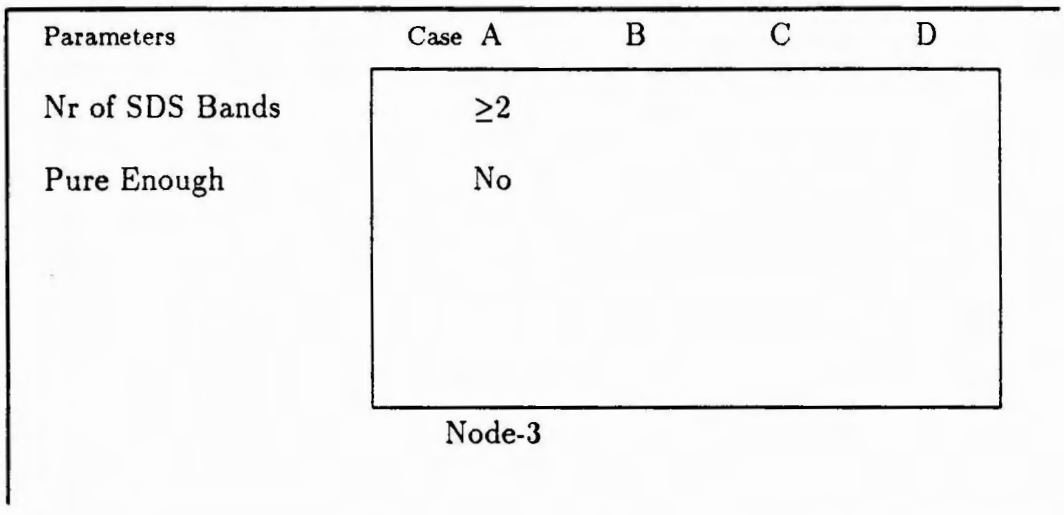


Figure 3: *A decision table filled in by the domain expert (adopted from P10).*

```
(defrule decision-table-1
      ((col :in (A B C D)))

      (and premise strategy-seq)

      (transform decision-table-rule
            "create rule {rule-name} in backward is"
            "{explain-dp-rule}"
            "if {premise* and} then"
            "assert {scope-var}.SelectionStrategy to
                              new sequence {strategy-seq*, };"
            "end;"))
```

Figure 4: *A simplified transformation rule.*

```
create rule Rule_61 in backward is
  Explain "This value is needed in the decision point after CIX
       purify in prototype dilute soluble protein in solutions.";
  if
    Prototype_5_Node_5.currentworld.NoOfSDSBand >= 2  and
    not Prototype_5_Node_5.currentworld.PureEnough
  then
    assert Prototype_5_Node_5.SelectionStrategy to
                        new sequence Prototype_5_Node_3;
end;
```

Figure 5: *A rule (*Rule_61*) generated by P10 (Epitool). This rule is the result of a transformation from the decision table in Figure 3 through the transformation rule in Figure 4 (*decision-table-1*).*

tool, in this case Epitool[6] [7]. When the rule schema is instantiated (by run-time variable substitution), the resulting Epitool rule represents the contents of one column in the decision table.

The result of applying the transformation **decision-table-1** to the decision table in Figure 3 is shown in Figure 5. Other transformation rules were used to infer the value of **rule-name**, **explain-dp-rule**, **premise**, **scope-var** and **strategy-seq**. The parameter names in the conclusion of **decision-table-1** where substituted by their (inferred) values to gain the final Epitool rule.

The rule in Figure 5, which originates from the previously mentioned decision table, essentially states that **if** the number of SDS bands in this node (n_5) is ≥ 2 *and* the protein is *not* pure enough **then** the next node to be considered should be number 3, i.e.

$$\text{SDSbands}(n_5) \geq 2 \wedge \neg\ \text{PureEnough}(n_5) \supset \text{SelectionStrategy}(n_5) = [3]$$

which expresses the same thing as the decision table in Figure 3.

P10 uses about forty transformation rules to transform a variety of knowledge module types into Epitool knowledge bases. Rules, classes, instances and (user) questions are generated through transformation rules similar to **decision-table-1**. Other (transformation) rules enables the knowledge base generator to deduce parameters similar to **rule-name**, **premise**, etc., (i.e. to generate sub-expressions). The DOTS system itself comprises about two hundred transformation rules for mapping specifications entered by knowledge engineers into target KA tools.

Transformation rules in P10 and DOTS are always used in a backward-chaining mode. The inference strategy involves exhaustive backward search to find and transform all the knowledge entered into appropriate knowledge base structures. The generation process is started by a query for the top-level target structure.

4.2.2 Starting with a form

This example illustrate the generation of rules from menus and forms filled in by experts. The form and an intermediate representation used are described as well as a transformation rule that map the representation into rules in the target knowledge base. (The form and the transformation rule used in this example is a simplified version of an actual case in P10.)

[6] Epitool is a trademark of Epitec AB.

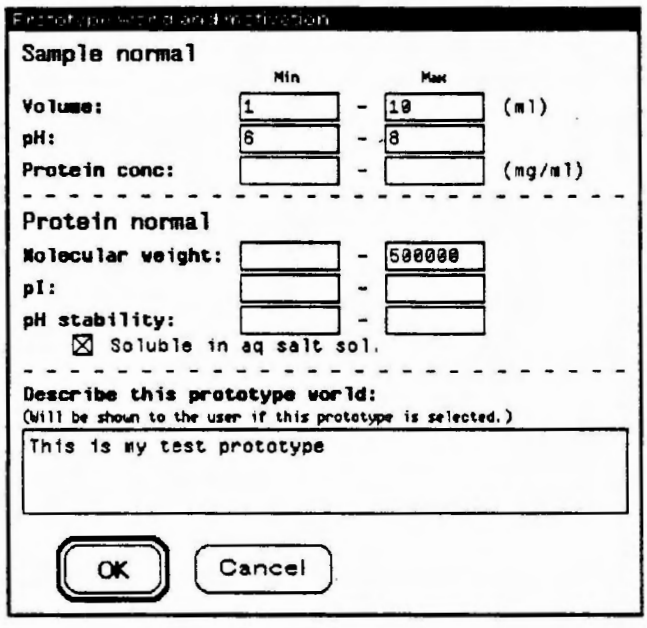


Figure 6: *A sample form that is filled in by domain experts. This form is actually used in P10 to acquire motivations from experts creating a prototypical solution [9].*

The sample form, which is specific to a biochemical domain, is shown in Figure 6. This form is used to acquire motivations from experts when creating new prototypical cases. Part of what is entered in this form will eventually be used as enabling conditions (which are represented as rules in the target knowledge base) for this prototypical case.

The contents of the form in Figure 6 state that the *sample volume* should be between 1 and 10 ml, the *sample pH value* should be between 6 and 8, and the *molecular weight* of the target protein should be less than 500,000 g/mol. Further the target protein should be soluble in aqua-salt solution. The description text shown in Figure 6 ("This is my...") is intended to be used in explanations to the end-user.

In P10, the contents of this menu (and similar ones) are transferred into an *intermediate* (object-oriented) representation by a simple copying mechanism, i.e. the values of various fields are copied into slots of instances. Table 1 shows the slots and their values (corresponding to what have entered in the form in Figure 6). A brief description of each slot is also provided in Table 1.

A set of (simplified) transformation rules that maps the intermediate representation into a target rule are shown in Figure 7. Rule **form-1** is used to generate the target rule and rules **form-premise-1** to **form-premise-3** are used to transform the slot values (Table 1) to a premise for the target rule. Rule **form-premise-1** states that the relation greater-than-or-equal (**>=**) should be used if only the minimum value is known (i.e. entered in the form in Figure 6) and rule **form-premise-2** states that less-than-or-equal (**>=**) should be used if only the maximum value is known. Rule **form-premise-3** states that a special range expression (*x* **is a real between** *min* **..** *max*) should be used of both the minimum and maximum values are known. Conclusions from rules **form-premise-1** to **form-premise-3** are collected on a list and used by rule **form-1** in the instantiation of the target rule (i.e. the target rule premise is generated through run-time variable substitution). If all values in the form are left blank, rule **form-1** will generate the premise **true**, i.e. there are no enabling conditions for the prototype in question. Note that additional transformation

Slot:	*Value:*	*Description:*
`volume-min`	1	Minimum sample volume
`volume-max`	10	Maximum sample volume
`pH-min`	6	Minimum pH value
`pH-max`	8	Maximum pH value
`protein-conc-min`		Minimum protein concentration
`protein-conc-max`		Maximum protein concentration
`mol-weight-min`		Minimum molecular weight
`mol-weight-max`	500,000	Maximum molecular weight
`pI-min`		Minimum pI value
`pI-max`		Maximum pI value
`pH-stability-min`		Minimum pH stability (range)
`pH-stability-max`		Maximum pH stability (range)
`sol-aq-salt-sol`		Soluble in aqua salt solution (boolean)
`description-text`		Description text (string)

Table 1: *Slots for a class representing values entered in the form shown in Figure 6. The values correspond to what is entered in the form (see Figure 6).*

rules are required to generate complete target rules (with rule names, explanations, etc.).

The result from this transformation, `Rule_15`, is shown in Figure 8. The rule in Figure 8 essentially state that the enabling condition for Prototype 1 is true if the volume is between 1 and 10, the pH value is between 6 and 8, and the molecular weight is less than or equal to 500,000, i.e.

$$(\text{volume(sample)} \in]1, 10[) \wedge (\text{pH(sample)} \in]6, 8[) \wedge$$
$$(\text{weight(protein)} \leq 500{,}000) \supset \text{PreconditionOK(Prototype_1)}$$

which express the same an the contents of the form in Figure 6.

Note that premise in `Rule_15` only includes slots that were non-nil in the intermediate representation (Table 1), i.e. fields that were non-empty in the form in Figure 6. Empty fields are removed to optimize the target rule. Also note that the order of the range expression in the premise can not be controlled since rules `form-premise-1` to `form-premise-3` may fire in any order. The transformation rules have to be rewritten if a specific order is required.[7]

4.3 Generating structures for explanations

When a core knowledge base is run together with parts generated, the user dialogue will be related to both the core and the knowledge generated. In a verification situation this can be confusing for a domain expert; knowledge entered merged with "built-in" knowledge results in unexpected, but not necessarily wrong, behaviour of the resulting system. The P10 system addresses this problem by allowing the user to select the level of built-in (i.e. core) knowledge at the beginning of a consultation. Switching off built-in knowledge is particularly useful when reviewing and debugging knowledge entered into P10. Why-explanations automatically created by the generator provide the user with means to determine the origin of particular questions (from the expert system to the end-user). A reference to the corresponding source in the knowledge acquisition tool is included

[7] Due to clarity of this example, a simple way of structuring the transformation rules was chosen.

```
(defrule form-premise-1
      ((s :in ((volume-min volume-max "sample.volume")
               (pH-min pH-max "sample.pH")
               (mol-weight-min mol-weight-max "protein.weight")
               (pI-min pI-max "protein.pI"))))

      (and (slot-value km (first s)) (not (slot-value km (second s))))

      (transform "{(third s)} >= {(first s)}"))


(defrule form-premise-2
      ((s :in ((volume-min volume-max "sample.volume")
               (pH-min pH-max "sample.pH")
               (mol-weight-min mol-weight-max "protein.weight")
               (pI-min pI-max "protein.pI"))))

      (and (not (slot-value km (first s))) (slot-value km (second s)))

      (transform "{(third s)} <= {(second s)}"))


(defrule form-premise-3
      ((s :in ((volume-min volume-max "sample.volume")
               (pH-min pH-max "sample.pH")
               (mol-weight-min mol-weight-max "protein.weight")
               (pI-min pI-max "protein.pI"))))

      (and (slot-value km (first s)) (slot-value km (second s)))

      (transform "({(third s)} is a real between {(first s)}..{(second s)})"))


(defrule form-1
      ()

      t

      (progn
        (setq form-premise (or (query form-premise :bag t) "true"))
        (transform form-rule
              "create rule {rule-name} in backward is"
              "{explain-form-rule}"
              "if {form-premise* and} then"
              "assert {prototype}.PreconditionOK to true;"
              "end;")))
```

Figure 7: *Transformation rules for generation of target rules for prototype preconditions. Rule* **form-1** *generates an Epitool rule that asserts* **PreconditionOK** *of the prototype to* **true** *if the premise is fulfilled. To determine the premise, rule* **form-1** *queries the variable* **form-premise**, *which in turn invokes rules* **form-premise-1** *through* form-premise-3. *Conclusions from these rules are collected on a list and used by* **form-1** *to generate the premise of the target rule.*

```
create rule Rule_15 in backward is
  Explain "This is needed in the precondition for prototype 1.";
  if
    (sample.volume  is a real between 1..10) and
    (sample.pH  is a real between 6..8) and
    protein.weight <= 500000
  then
    assert Prototype_1.PreconditionOK to true;
end;
```

Figure 8: *The resulting rule (*`Rule_15`*) from the form in Figure 6 (Epitool). This rule was generated by the transformation rules shown in Figure 7 (and some other transformation rules not shown here).*

in explanations for rules generated. The "explain" statements in `Rule_61` and `Rule_15` are used for the why-explanations (see Figures 5 and 8 respectively). SALT is another example of a KA tool that incorporates explanations [18, Section 5]. Expert systems generated from SALT can explain their decisions using records of the problem-solver's use of the domain knowledge.

4.4 Alternative approaches

Approaches other than knowledge base generation to gaining an executable system are possible, for instance utilizing call-out[8] from a shell, generating internal shell structures,[9] or interpreting the contents in the knowledge acquisition tool directly. However, these approaches were not considered realistic in the P10 and DOTS projects, mainly because of the lack of expressiveness, maintainability and suitability of meta-tool generation. This was also the motivation for choosing transformation rules as a basis for knowledge base generation.

Code generation in compilers [17] is an issue related to the generation of knowledge bases from knowledge acquisition tools. A number of approaches to generating efficient machine code from intermediate representations in the compiler have been suggested and implemented, for instance *tree pattern matching* and *table-driven techniques.*

Although generation of machine code seems to be similar to knowledge base generation, there are some fundamental differences. Compilers generate data structures and procedural instructions, whereas knowledge acquisition tools typically generate declarative knowledge bases. In the compiler case, the starting point is often an abstract syntax tree. For a general knowledge base generator this starting point is harder to define, since the input consists of more or less arbitrary representations of elicited knowledge structures. Requirements for error handling and generation of explanations exemplify another difference. The run-time library of a compiler roughly corresponds to core knowledge bases (see Section 3) in the approach taken.

The motivation for choosing the rule format described here in the P10 and DOTS projects was mainly clarity of the rules. The transformation rules are specified in terms

[8] Many shells provide facilities for calling external functions (written in, for instance, Lisp). A KA environment could generate such functions. Alternatively, external functions could provide an entry point to the KA tool.

[9] Internal knowledge base (data) structures of the shell can potentially be generated. However, this approach requires access to and documentation for these internal structures. Structures generated would also depend on the shell version.

of the syntax used in internal knowledge representations and in target knowledge bases. Note that this is actually a form of syntactic sugar (which improves clarity) since such transformation rules can still be used to express transformations on a deep (i.e. semantic) level.

One could consider using transformation rules on even deeper levels. In such an approach, fundamentally different representations could potentially be transformed into each other and the knowledge structures could be optimized according to certain criteria, e.g. inference speed or storage space. However, using such extensive transformations for knowledge base generation can result in control knowledge in the target inference engine being moved into the transformation rules of the knowledge base generator. This can be undesirable, since it contradicts the aim of separation of knowledge from inference. It is reasonable to take an approach where transformation rules for knowledge base generation only represent the mapping from internal representations to the target knowledge base format. The refinements mentioned above should be performed either in the knowledge acquisition tool (as part of its elicitation mechanism) or in the generated knowledge base, e.g. by knowledge base transformation [23]. Examples of such knowledge base tuning systems are Refine[10] [25; 22] and SEEK2 [12].

5 Multiple target formats

Generation of knowledge bases in different formats, e.g. KEE[11], CSRL [4], SOAR [15], Epitool, Nexpert[12] or C++, is an important requirement on a general knowledge acquisition system. The ability to generate multiple formats can be found in many existing tools, for instance AQUINAS [2], and facilities for generation of knowledge bases in conventional languages[13] can be found in tools such as KNAPS [16]. In compiler construction, *retargeting* has been an important design consideration for a long time [26; 1].

In domain- and generic-task-specific tools, implementation of support for multiple formats will be laborious compared to the benefits it realizes for a small user group community. This could be remedied by introducing an intermediate stage in the knowledge base generation process. As shown in Figure 9, knowledge structures from the tool are first transformed into sets of target-independent ***knowledge-base units***[14] (U_i). The generation process is finalized through sets of target-dependent transformation rules (T_i^1). A different format can be generated through T_i^j. The sets T_i can be reused in other KA tools possibly specialized to other domains/generic tasks. Retargeting to another knowledge base format will only require the target-dependent transformation rules to be rewritten for the new format. Transformations to knowledge-base units will be unaffected or only marginally affected. An alternative for utilizing transformation rules in the final phase could be to guide the finalization by an abstract syntax representation of the target format, i.e. a specification of the target language.

The feasibility of the knowledge base generator (for multiple formats) outlined is critically dependent on the ability to represent target structures, i.e. knowledge-base units. A reasonable approach to this problem is to restrict the units to the most common knowledge base entities and to only generate forms corresponding to the units in a conventional

[10] Refine is a trademark of Reasoning Systems, Incorporated.

[11] KEE is a trademark of IntelliCorp, Incorporated.

[12] Nexpert is a trademark of Neuton Data, Incorporated.

[13] Doyle [6] argues that the major advantage of AI programs (e.g. expert systems) is the development approach rather than the symbolic reasoning employed in such systems. Conventional implementation (coding) techniques are often a better choice.

[14] The term "knowledge-base unit" is used to denote a structural unit in a target-independent format, e.g. class, instance or rule.

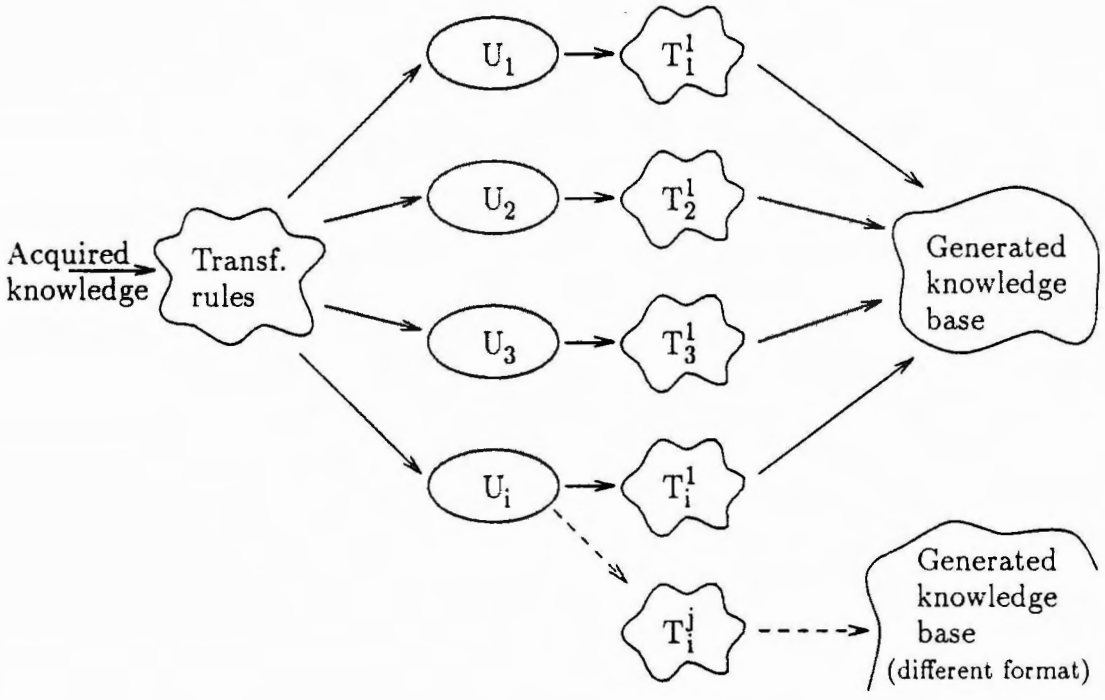


Figure 9: *Target-independent knowledge-base units as an intermediate stage in knowledge base generation.*

language [3]. Another problem is to find suitable representations for expressions in rule premises etc.

Knowledge base generators in KA tools generated by DOTS as well as the KA tool generator in DOTS itself[15] support multiple target formats. The DOTS system uses transformation rules based on the rule format described in Section 4 to transform internal representations into knowledge-base units. In the finalization phase, *procedural* transformation rules [21, Section 1.2.4] are used to transform knowledge-base units into the appropriate constructs in the target knowledge base format.

To introduce a new target knowledge-base format in DOTS these procedural transformation rules need to be rewritten. Target formats are structured hierarchically and these rules are implemented as methods which can be inherited from more general formats. By using this organization of target formats, only the differences in new versions of inference engines need new rules. For instance, if implementors of hybrid tools release new versions, support for *all* versions can more easily be implemented. Examples of procedural rules specific to the target format include

add-code, which generates target constructs for *adding* structures corresponding to new knowledge-base units;

delete-code, which generates target constructs for *deleting* structures corresponding to deleted knowledge-base units;

change-code, which generates target constructs for *replacing* obsolete structures with updated ones from new versions of knowledge-base units; and

target-name, which returns legal names (in the target environment) for knowledge-base units.

[15]The generation mechanism is part of the *common library* in the DOTS system [9].

Retargeting to a new knowledge base format can be achieved by rewriting rules similar to the ones mentioned above.

The procedural rules found in the DOTS system do not cover all aspects of the target knowledge base formats supported. More sophisticated transformations are certainly needed in DOTS. In particular, the procedural rules in the current version of DOTS can not transform arbitrary expressions (e.g. rule premises) into different target formats. However, knowledge engineers can declare in which target formats a specific knowledge-base unit is valid.

6 Incremental regeneration

Incremental regeneration of knowledge bases can significantly improve the turnaround time in the knowledge acquisition and test-run loop in expert system development, because only changes since the latest knowledge base generation are generated (and subsequently loaded into the target environment) in this approach.

6.1 Motivation

It is often desirable to give domain experts rapid feedback of knowledge elicited in terms of running successive versions of the target system. Currently the bottleneck in the process of gaining executable systems from KA tools is *not* the generation time, but the time it takes to load the knowledge generated base into a hybrid shell.[16] This is primarily caused by internal syntactic and semantic checking as well as rule indexing in the shell. Consequently, most shells offer an incremental update of the knowledge base. Incremental update of the target knowledge base (i.e. generate changes only) is one way of exploiting incrementality of the shell and, thus, removing this bottleneck. Incremental regeneration, on the other hand, will require a more sophisticated knowledge base generator to enable knowledge structures generated to be retracted.

6.2 Knowledge modules

In order to achieve incremental update, the elicited knowledge must be separated in fairly independent parts, which are the minimal changes detected by the knowledge base generator. A reasonable size for these *knowledge chunks* could be a small set of rules, e.g. a decision table. The knowledge acquisition tool may create, modify and delete such knowledge chunks, which in turn are the input to the knowledge base generator.

In P10 and DOTS, such knowledge chunks are equivalent to knowledge modules [8; 9], which are implemented in an object-oriented language as instances of the knowledge module class or one of its sub-classes. Knowledge modules are used for several different purposes in P10 and DOTS, including the representation of graphs, decision tables, forms, etc., acquired. Knowledge modules are atomic with respect to the incremental knowledge base generator in DOTS, i.e. one knowledge module modified is the smallest possible change. Naturally, internal representations other than knowledge modules are possible. However, there must be some form of knowledge chunks that defines the smallest change possible.

6.3 Detecting changes

Incrementality in the knowledge base generation process requires that the knowledge acquisition system keep track of new knowledge modules, change to existing knowledge

[16] In the P10 project, it took typically 3–5 times longer time to load knowledge bases into environments such as KEE or Epitool than to generate them.

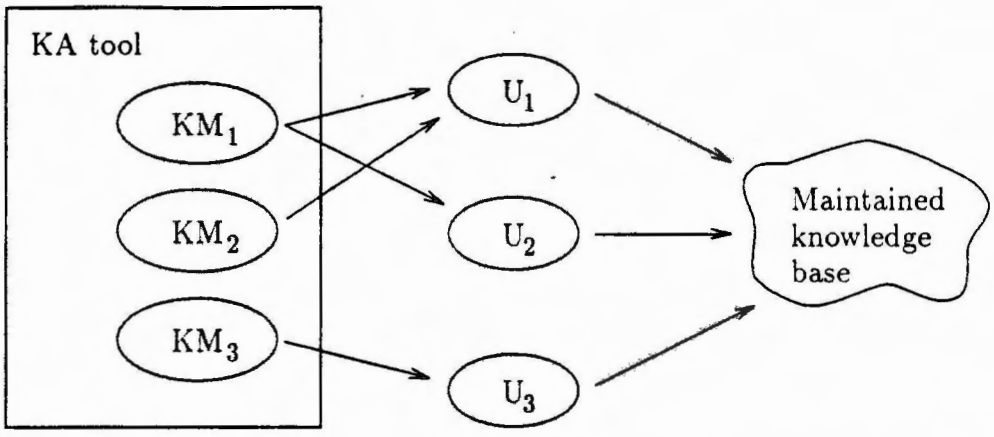


Figure 10: *Utilization of a dependency network for incremental knowledge-base generation.*

modules and delete of knowledge modules. The status of previously generated knowledge-base units (Section 5) can be determined by maintaining a dependency network which describes the origin of each knowledge-base unit (see Figure 10). A status change in a knowledge module can be propagated to the knowledge-base units affected immediately after the modification occurred or at generation time.

The meta-tool maintains the set of knowledge modules affected for future use in knowledge-base regeneration. When incremental generation is started, the meta-tool regenerates knowledge-base units for each of the knowledge modules in the affected set. This process can, for instance, be guided by transformation rules.

If a knowledge module has been marked as *added,* the resulting knowledge-base units are used directly. On the other hand, if it has been marked as *changed,* the new knowledge-base units are *unified* with the old ones. This means that the new and old ones are tested to determine equality with respect to the target language/shell. If the new and old ones *do* unify, no action is taken, otherwise the old one is retracted and the new one added to the target knowledge base. Knowledge modules marked as *deleted* do not require regeneration of any knowledge-base units. If the old knowledge-base unit only depends on this deleted knowledge module (with respect to the dependency network), it is retracted from the target knowledge base, i.e. instructions for removing corresponding structures in the target knowledge base are generated.

6.4 Finalizing and effecting

The finalization of these units involves the generation of instructions for modification and deletion of target structures, e.g. "undo" of previously generated structures. DOTS has procedural rules (or methods) for finalizing *added, changed* and *deleted* knowledge-base units (i.e. generating the syntax) for each of the target languages supported.[17]

Since knowledge-base generation and loading of structures generated into target environments are time-consuming processes, time can be saved by pipelining newly generated structures directly to a target machine. In this way, the target machine can start loading the finalized knowledge-base unit as soon as it is ready. This implies that the generator itself must be pipelined internally, i.e. knowledge-base units must be generated, finalized and sent to the target for one knowledge module at a time. (The knowledge base generator in DOTS has facilities for pipelining.)

[17] New target languages can be introduced by adding more methods.

Target shells usually require the knowledge-base units to be (partially) ordered, e.g. a class must be defined before its methods and instances. Such ordering can be achieved by sorting the knowledge-base units generated afterwards (utilized in P10). This strategy can, however, not be applied in a pipelined generator. DOTS applies heuristics in this case by sorting the knowledge modules (with respect to a priority number/weight given by the knowledge engineer) as a prestep and afterwards sorting the resulting knowledge-base units from each knowledge-base unit (according to target requirements).

It may be impossible or very expensive to retract basic components of target knowledge bases, such as global declarations. In such cases the entire knowledge base must be regenerated/reloaded. If pipelining is used, it may be difficult to determine if a forthcoming basic component will changed/retracted and, thus, require a complete regeneration of the knowledge base. Again DOTS uses heuristics to determine if a complete regeneration is needed. Knowledge modules causing basic components to be generated can be marked as critical by knowledge engineers in advance.

7 Integrating generated output

Integrating generated knowledge bases with core knowledge bases (or any other type of knowledge base) is a difficult problem. It is important to keep in mind that today's commercial shells and hybrid tools do *not* have any particular mechanisms for this type of knowledge base integration and are typically ill-suited for this task. Further, surprisingly little has been reported on this issue and generic knowledge bases are still at an early stage in development. However, in the mean time there are a few useful "tricks" to achieving a reasonable level of integration between generated and core knowledge bases. A fruitful approach is to utilize *specialization* in object-oriented programming. (Typically, a hybrid tool or a custom-made inference engine are assumed.)

As shown in Figure 11, root and near-root classes in the taxonomy should be defined in the core knowledge base (A–D). The KA tool can then generate specializations (i.e. subclasses and instances) to these classes (E–J). Generic-task-specific slots should be defined in classes related to the generic-task-specific part of the core knowledge base, whereas domain-specific slots should be defined in the domain-specific part (see Section 3). By defining additional slots in (generated) subclasses it is possible to introduce new slots generated from the KA tool. Thus, it is possible generic-task-oriented, domain-oriented (manually encoded) and generated slots in the class hierarchy of the final knowledge base.

When it comes to integration of rules additional difficulties are encountered. Typically, KA tools generate a number of predefined *rule types,* i.e. rules of similar format intended to be used for specific tasks in the problem solving method. For example, SALT produces six main rule types [18] and KNACK produces five different rule types[18] [14]. Such rule types can be *complete* in the sense that the set of rules covers all the possible cases. If rules are generated from a decision table (or any other form of classification matrix), every possible case may be covered. If not, a default behaviour can be specified in the core knowledge base, or alternatively, generated from the KA tool. For set of rules which cover all cases it is typically not meaningful add more rules.

In other situations it can be desirable to add more rules to a pre-existing rule set. A few rules (in the core knowledge base) can constitute a simple default behaviour and additional rules generated from the KA tool can improve performance by adding further expert knowledge. If rules are used to infer slot values of instances (e.g. in a hybrid tool),

[18] That is, identify-relevant-information, define-a-strategy, integrate-information, define-the-skeletal-report and define-a-fragment rules [14, Section 5.5].

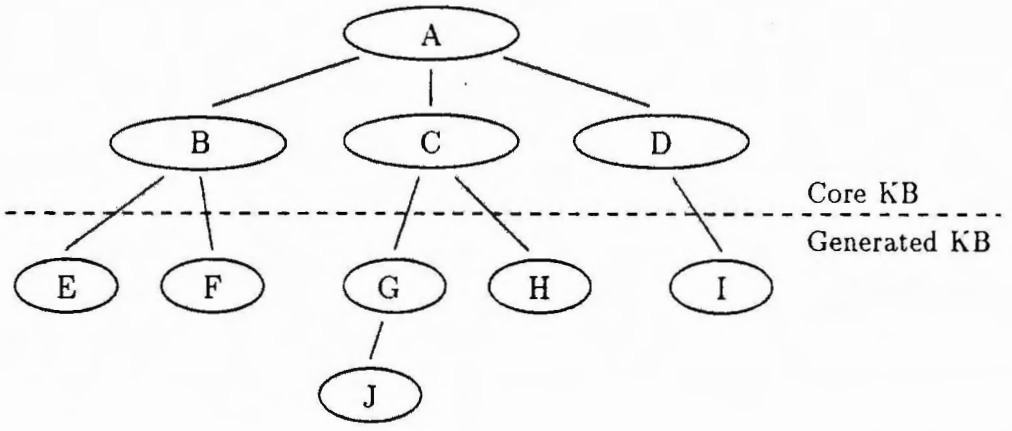


Figure 11: *Integration of classes in the core and the generated knowledge base (KB). The most general classes should be defined in the core knowledge base (A–D) and specializations to them might be generated from the KA tool (E–J).*

the rules can be hooked up to the inheritance hierarchy. Rules in the core knowledge base can assert values in the top-most classes and generated rules can assert values (possibly for the same slots) of subclasses generated.

8 Summary and discussion

Architectures for generic knowledge acquisition tools (including generic transformation and knowledge base generation mechanisms) are primarily motivated by the construction of meta-level tools (i.e. tools for customization of knowledge acquisition environments), but are also of general interest, e.g. in manual implementation. Generic generators based on transformation rules is an important step towards domain- and generic-task-independent meta-tools [8; 9]. The separation of the target knowledge base into generated and core parts allows the knowledge engineers to mix manually-acquired knowledge with tool-elicited structures. As described in Section 3, rare and exceptional cases can be stored in the domain-specific part of the core knowledge base, whereas generic task-oriented knowledge should reside in the generic-task-specific section. Generation of subclasses and instances to classes in the core knowledge base is an approach to the problem of integrating the output with the core knowledge base (Section 7). Transformation rules has successfully been employed to implement knowledge base generators for a number of knowledge acquisition tools (Section 4). This approach provide several advantages including clarity of the transformer, improved maintainability, possible support for multiple target formats, and—if required—incremental update.

By developing KA tools in a manner where implementation of the transformation process is delayed until the final stages of tool development, much of the technical details in the transformation process can be abstracted during stages where expert participation is required (in the KA tool development). Moreover, flexible knowledge base generators make it possible to change the implication of already elicited knowledge by changing the transformation rules. Thus, it is possible to afterwards compensate for misunderstandings and flaws in the conceptual model of knowledge already acquired by the KA tool. Multiple target formats (Section 5) can contribute to the strategy of postponing decisions regarding the target environment for knowledge bases generated. If developers strive to attain an incremental development style, efficient regeneration of knowledge bases will

eventually be required. Hence, incremental regeneration (Section 6) is an important aspects of a knowledge base generator intended for a knowledge acquisition "production" environment. Integration of generated output with manually encoded knowledge bases is a difficult problem subject to further research. Currently, one of the best approaches is to use specialization in object-oriented programming. However, specialization in other dimensions then merely classes, i.e. datatypes, is highly required. Further limitations on this approach include that (1) transformation rules are unsuitable for expressing some types of transformations (e.g. parsing), (2) multiple target formats can not easily be made transparent for all types of inference engines, and (3) complexity problems may arise in incremental regeneration. (However, heuristics can be used to reduce the complexity problem.)

Acknowledgments

Thanks are due to Sture Hägglund, Peter Eklund, and Kristian Sandahl for valuable discussions and comments on drafted versions of this paper.

References

[1] Alfred V. Aho, Ravi Sethi, and Jeffery D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, Reading, Massachusetts, 1986.

[2] John H. Boose and Jeffery M. Bradshaw. 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, 1987.

[3] Charles W. Butler, Earl D. Hodil, and Gary L. Richardson. Building knowledge-based systems with procedural languages. *IEEE Expert*, 3(2):47–59, 1988.

[4] Tom Bylander and Sanjay Mittal. CRSL: a language for classificatory problem solving and uncertainty handling. *AI Magazine*, pages 66–77, August 1986.

[5] B. Chandrasekaran. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert*, 1(3):23–30, 1986.

[6] Jon Doyle. Expert systems and the "myth" of symbolic reasoning. *IEEE Transactions on Software Engineering*, SE-11(11):1386–1390, November 1985.

[7] Epitec AB, Linköping, Sweden. *Epitool Development Environment Reference Manual*, 1989. (Version 4).

[8] Henrik Eriksson. Meta-tool support for customized domain-oriented knowledge acquisition. In John H. Boose and Brian R. Gaines, editors, *Proc. of the fifth Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, pages 6.1–6.20, Banff, Canada, November 1990.

[9] Henrik Eriksson. *Meta-Tool Support for Knowledge Acquisition.* PhD thesis 244, Linköping University, 1991.

[10] Larry Eshelman. MOLE: A knowledge acquisition tool that buries certainty factors. *International Journal of Man-Machine Studies*, 29(5):563–577, 1988.

[11] Martin S. Feather. A survey and classification of some program transformation approaches and techniques. In L. G. L. T. Meertens, editor, *Program Specification and Transformation*, pages 165–195. North-Holland, 1987.

[12] Allen Ginsberg and Sholom Weiss. SEEK2: a generalized approach to automatic knowledge base refinement. In *Proc. of the Ninth International Joint Conference on Artificial Intelligence, IJCAI'85*, pages 367–374, 1985.

[13] Catherine M. Kitto and John H. Boose. Heuristics for expertise transfer: An implementation of a dialogue for knowledge acquisition. *International Journal of Man-Machine Studies*, 27(2):183–202, 1987.

[14] Georg Klinker. KNACK: sample-driven knowledge acquisition for reporting systems. In Sandra Marcus, editor, *Automating Knowledge Acquisition for Expert Systems*, chapter 5, pages 125–174. Kluwer Academic Publishers, Norwell, Massachusetts, 1988.

[15] John E. Laird, Allen Newell, and Paul S. Rosenbloom. SOAR: an architecture for general intelligence. *Artificial Intelligence*, 33:1–64, 1987.

[16] Newton S. Lee. Graphical knowledge programming with KNAPS. *International Journal of Man-Machine Studies*, 31(6):611–641, 1989.

[17] Hans Lunell. *Code Generator Writing Systems.* PhD thesis 94, Linköping University, 1983.

[18] Sandra Marcus and John McDermott. SALT: a knowledge acquisition language for propose-and-revise systems. *Artificial Intelligence*, 39:1–37, 1989.

[19] John McDermott. Preliminary steps toward a taxonomy of problem-solving methods. In Sandra Marcus, editor, *Automating Knowledge Acquisition for Expert Systems*, chapter 8, pages 225–256. Kluwer Academic Publishers, Norwell, Massachusetts, 1988.

[20] Mark A. Musen. An editor for the conceptual models of interactive knowledge-acquisition tools. *International Journal of Man-Machine Studies*, 31(6):673–698, 1989.

[21] H. Partsch and R. Steinbrüggen. Program transformation systems. *Computing Surveys*, 15(3):199–236, September 1983.

[22] Reasoning Systems Inc., Palo Alto, CA. *Refine User's Guide*, 1987.

[23] Peter Rothman. Knowledge transformation. *AI Expert*, 3(11):28–35, November 1988.

[24] Nosrat Shahsavar. *Knowledge Acquisition and Refinement for a Domain-Specific Expert System.* Linköping Studies in Science and Technology, Licentiate thesis 229, LiU-Tek-Lic 1990:29, ISBN 91-7870-671-8, Linköping University, 1990.

[25] Douglas R. Smith, Gordon B. Kotik, and Stephen J. Westfold. Research on knowledge-based software environments at Krestel institute. *IEEE Transactions on Software Engineering*, SE-11(11):1278–1295, November 1985.

[26] J. Strong, J. Olsztyn, J. Wegstein, O. Mock, A. Tritter, and T. Steel. The problem of programming communication with changing machines—a proposed solution. *Communications of the ACM*, 1(8):12–18, 1958. Part two in 1(9):9–15.

CATO - An Automated Card Sort Tool

Nigel Major*
Artificial Intelligence Group
Department of Psychology
University of Nottingham
Nottingham NG7 2RD
England
email: nigel@psyc.nott.ac.uk

Abstract

This paper presents CATO, a computer-based card-sort tool. CATO is an implementation of the card sort 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, the technique of card sorting is introduced and an implementation described. One of the more interesting aspects of CATO is its ability to analyse the knowledge that has been elicited, and also to perform automatic reporting of that knowledge. Some of the flexibility needed by users is discussed, together with a transformation of the knowledge into first-order predicate calculus, and general implications that our work on CATO has thrown up for the design of a knowledge engineering workbench.

1 Introduction

One of the most difficult aspects of knowledge engineering is the task of eliciting knowledge from an expert. This task is particularly difficult because experts may find it hard to formulate their knowledge explicitly. To combat this, a number of techniques have been devised to maximise the yield of usable knowledge obtained from experts. Broadly, these fall into two categories, standard techniques and "contrived" techniques [3]. Standard techniques involve either interviews, structured ([6], [5]) or unstructured [14], or descriptions of the expert's task, produced either by the expert as the task is performed, or by one expert watching another ([8], [4]).

*The research reported here was performed as part of the ESPRIT-funded ACKnowledge project. Colin Corbridge and Gordon Rugg informally evaluated an earlier version of CATO, and provided valuable feedback which has substantially improved the tool.

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 analyses on the knowledge. Examples of tools implementing contrived techniques include laddering [10], repertory grid analysis [13] and the limited information task [7]. Card sorting is another example. This paper describes, CATO, a card sorting tool, written in Common Lisp.

CATO 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 [11].

The organisation of the paper is as follows. In section 2, card sorting is discussed in more detail. In section 3, the current state of the card sort tool is described. In section 4, an example of CATO in operation is given. In section 5, some of the implications of the work for the design of a KEW are discussed. Section 6 finally concludes and discusses further work.

2 The card sort technique

The card sort technique has been shown to differ significantly from the repertory grid technique. The most notable is with regard to categorical knowledge, where the knowledge does not fall easily along a continuous dimension, for example colour names. To use a repertory grid in this case would impose a structure on the knowledge which is really not present. Card sorting is particular suitable however for this kind of problem. Consequently any knowledge engineering workbench would be well justified in including both techniques.

Traditional card sorting involves a set of cards (similar techniques apply to sorting items or photographs, but only cards are discussed here), each of which represents a particular item in the expert's domain. The items represented should be at the same knowledge level in the domain, so meaningful sorts can be performed. For example, in the domain of ships, it would be difficult (although not impossible) to sort cards representing "QE2" and "Mizzen mast".

An initial set of cards is chosen, either from scratch or as the output of some other knowledge acquisition tool. The expert is then allowed to add or remove cards as they wish. It is important that cards are not added or removed after sorting has started. The expert is then asked to sort the cards into groups according to some criterion, with the groups being different values of the criterion. So if the sort were

according to size, the groups might be "small", "medium" and "large". It is a good idea to allow a group of "don't know" or "not applicable", as otherwise the knowledge elicited will be restricted to that which is known by the expert with respect to all the cards. This could mean quite a loss of useful knowledge. Repertory grids have difficulty handling this area of knowledge restricted to only some of the concepts being considered, in that they can only use constructs whose range of convenience covers all concepts ([9], [2]). Once all the cards have been put into piles, the sort is recorded.

If the expert has difficulty in thinking of new criteria, then prompts may be given for example by showing the expert three cards, and asking which is the odd one out and why. This is the triadic elicitation method used in repertory grids. In practice, however, this is seldom necessary.

The process continues until no more criteria can be thought of.

One other possibility is to supply criteria to the expert in order to pool knowledge from different experts. This is particularly useful in domains where there may be conflicting knowledge.

Along with other contrived techniques, it has been shown that card sorting is an extremely efficient knowledge elicitation technique, with a high productivity in terms of knowledge elicited versus length of time of elicitation [3].

3 CATO

CATO was originally implemented in SWI-Prolog using the PCE user interface development environment [1]. Since the original implementation, PCE has been ported to run with Sun Common Lisp and so CATO was also ported to this environment, with significant improvements.

CATO asks the user a few initial questions about the domain and also what the concepts are that are to be sorted, and from then on the control is very much in the hands of the user. As concepts are named, cards appear on the display, and these can be added to or removed at any time until the first sort has been completed. Once the concepts are named, sorting can begin.

The user has great flexibility in the way in which sorts are made. For example the cards may be sorted first without any mention of the criterion used for sorting or the groups intended. This is particularly useful when the expert is finding it difficult to think of sorts and just wants to play around with the cards for a while to help think of ideas. Once the cards have been sorted the groups are given by the user and the criterion for the sort named. This updates the display so that the cards in different groups are separated and labelled. When the expert is happy that the sort is correct the sort is explicitly accepted. Then CATO reads the sort from the display, generates rules from the information, and stores them. The sort can be cancelled up at any stage to this final acceptance.

If the expert has sorts in mind immediately then the groups and the sort crite-

rion can be given before sorting, all that is then requested is that the expert arrange the cards into the groups on the display. If the sort is already in mind, then this is a lot easier than playing about with the cards before the groups have been named, as it is immediately apparent where each one should go.

As sorting continues, the expert may review the results so far in many different ways. CATO can show the criteria that have previously been used. This is the simplest form of reminder to the expert of the sorts so far used. Alternatively the rules generated can be displayed. These rules are an easy-to-read version of the rules are obtained by transforming the results of the sorts into logic (see section 3.2). They may also be interrogated by the user, which serves to list all the groups any particular card has been placed in, or which cards have been placed in a particular group. The most comprehensive display of results can be given by selecting the "Report" option. This shows the whole matrix of sorts against concepts, with the group of each concept in each sort. This report is also saved to a file. Although a relatively simple task, this provides one of the most useful features of automating the card sort, as it obviates the need for laborious collation of the results after a manual elicitation session. So overall, the expert has access to the card sort results in many different forms, which could be compared to the flexibility of a query language, without the requirement to learn any syntax.

The final feature that may be used during the actual sorting itself, is a form of knowledge analysis (see section 3.1). This informs the expert which concepts have been uniquely discriminated by the sorts so far, and which collections of concepts are not explicitly discriminated (all concepts are of course implicitly discriminated just by their being different cards). This may act as a prompt for either a knowledge engineer or the expert, as to further areas for sort criteria. This analysis can be compared closely to that performed by the ALTO laddering tool [10].

Figure 1 shows the interface of CATO, while working on the domain of car classification.

3.1 Knowledge analysis and reporting

CATO goes beyond the basic features of performing a card sort, in two main ways. First, it has the ability to analyse its own knowledge to provide cues for the user as to which concepts to concentrate on next, and it also supports useful reporting facilities for knowledge engineers still using paper and pencil techniques.

The knowledge analysis CATO uses is one of discrimination between the concepts which are being sorted. When requested to analyse the sorts so far, CATO takes each card in turn, and searches through the other cards to see if they have been placed in a different group for any of the sorts already done. If they have, then they have been discriminated from the card in question. If however they have the same group values for all the previous sorts, then they are indiscriminable from the card. This is despite the fact that they are different concepts which suggests that they should be discriminable. When this process is finished CATO reports all the

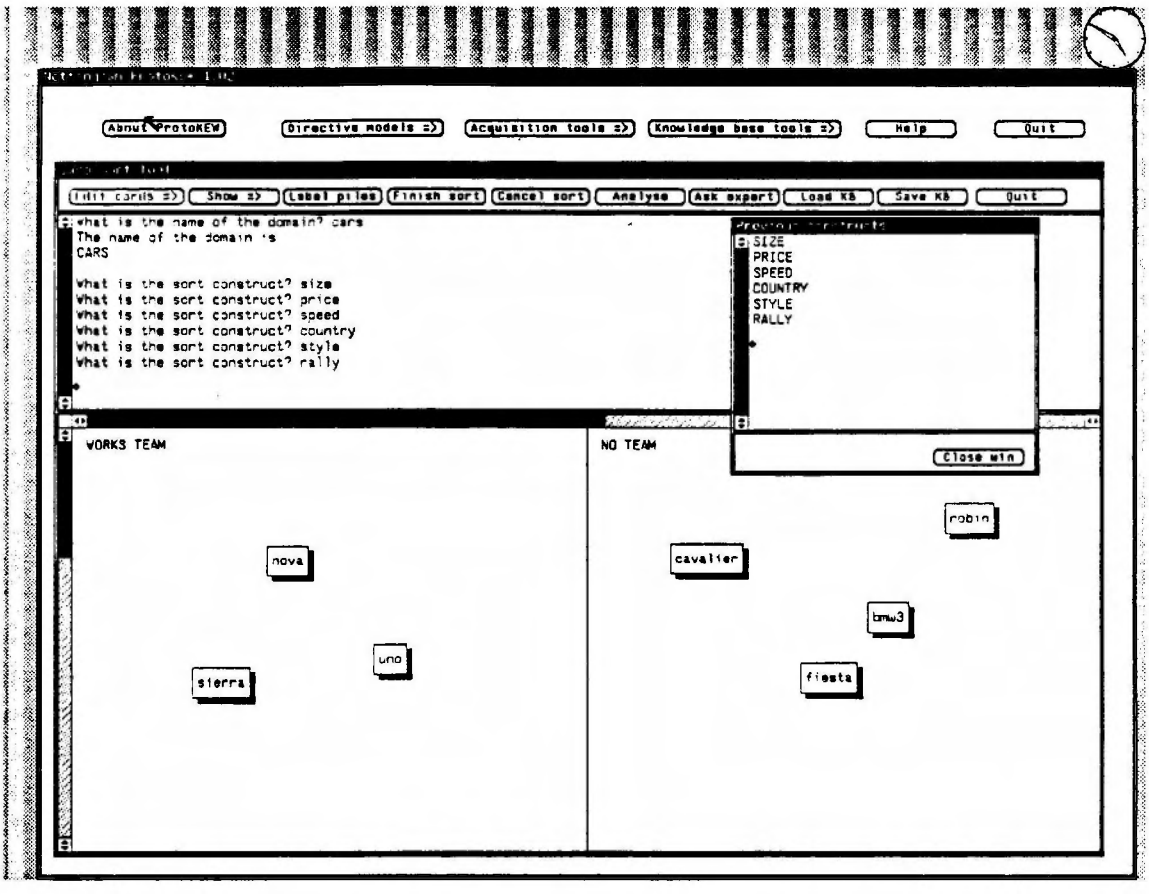


Figure 1: CATO working on the car classification domain

concepts that have been uniquely discriminated, and further, all the sets of concepts which still remain the same. This acts as a pointer to the user that the differences between concepts in such a set, should be used as criteria for future sorts. Obviously the aim here is to ensure that all concepts are uniquely discriminated.

A question for extending this analysis would be the degree of discrimination required by CATO from the expert. It could be that a knowledge engineer who was working alongside the expert as they used CATO, would want to ensure that discrimination was not made on the basis of one sort being different only, but that several sorts would be required with different values to discriminate two concepts. Further, with regard to including CATO in a knowledge engineering workbench comprising many tools (see section 6), those concepts which have been weakly discriminated, or indeed not discriminated at all, would be prime candidates for further elicitation with another technique. One might even expect such a workbench to spot this area of weakness and to ask the expert whether it is suitable for further elicitation using some other technique, or whether it is just not very richly described.

The second main feature over and above basic card sorting is CATO's reporting ability. When systems which have traditionally been carried out with paper and

pencil are automated, a crucial factor in user acceptance can often be reporting of results in a manner similar to that in which results have been reported previously. To this end, CATO is able to produce output, in a file, which mimics the result format used by knowledge engineers when performing manual elicitation. Indeed, this reporting was originally implemented independently of CATO, purely to speed up the result process for manual elicitation. Later, it was included in CATO as a very useful means of recording results. The matrix which shows all the sorts and values for each card is presented in the report, together with a list of all the groups for each sort. These are the crucial pieces of information used by knowledge engineers when processing card sort results.

3.2 Transformation into logic

CATO has the ability to transform its knowledge into logical propositions, using first-order predicate calculus. The primary reason for including a transformation is to integrate the knowledge with that from other knowledge acquisition tools. Logic can be formally checked for consistency and provides a convenient means of integrating knowledge from different sources. The transformation is an interactive process, and so acts as an extension to the elicitation. For example, the knowledge engineer is able to specify whether each card should be treated as an instance or a class, thus providing more information than would usually be found from the card sort alone. It is likely to be more effective for this purpose than a similar transformation into frames (see section 5).

The transformation of CATO's knowledge is done in the following way. For every concept *con*, in category *cat*, under criterion *crit*, with *othercon_i* other concepts, it asserts

$(\exists x)\ (con(x))$
$(\exists x)\ (cat(x))$

For classes, the following clauses are asserted.

$(\forall x)\ (con(x) \Rightarrow crit(x)(cat))$
$(\forall x)\ (con(x) \Rightarrow \neg othercon_i(x))\ (\forall i)$

Finally for instances, these clauses are asserted.

$crit(con)(cat)$
$\neg othercon_i(con)\ (\forall i)$

3.3 Evaluation

The tool has been informally evaluated by asking a small number of knowledge engineers, some of whom knew the card sort technique and some of whom did not, and also domain experts, to use the tool for elicitation.

The process of card sorting is very intuitive and easy to understand for experts. Consequently any tool which implements this technique should ensure that the user is always in direct control of what is happening. This meant that the requirements

of users at different levels of experience and familiarity with card sorts, did not differ very much. The knowledge engineers were more interested in the availability of reporting facilities than the experts, but this is understandable as it is they who would have to perform the reporting themselves if the tool did not provide it. The general requirement was for flexibility in the order of use of the tool, so that sorts could be done before or after groups were named and criteria given. Generally experts found the tool intuitive and simple to use.

3.3.1 A formal evaluation

Further to this, CATO has been part of a formal evaluation of sorting techniques. This study has carried out a formal comparison involving an item sort, a card sort and a computerised card sort using a slightly earlier version of CATO. The domain was that of fruit identification, and the purpose was to determine if sorting real fruits led to different types of knowledge elicited than sorting labels on a computer screen. The method was a between-groups design, with three groups of 25 people, each group doing a different type of sorting. Each subject performed half an hour of elicitation using the specified technique. Sort criteria were further classified into categories: physical properties; uses; personal preferences; linguistic and other.

The results showed that there was very little difference between the sort of knowledge elicited using CATO, and that obtained using the traditional manual technique. Thus the two versions of the card sort, manual and automated, were shown to be essentially equivalent. When coupled with the increased speed in reporting given by CATO, and the reduction in likelihood of errors, we may conclude that this automation is definitely a step forward for knowledge acquisition using card sorts.

4 Example of CATO in use

To give the reader a flavour of CATO and its different facilities, a short example is given. We will use the domain of cars for familiarity and for simplicity. Much of CATO's interaction is done by direct manipulation and so will not be represented here.

On starting up, CATO asks the user for the type of objects to be sorted, in this case cars, and then asks for names of cars to be sorted. As each name is given, a new car appears on the screen. When the user is satisfied that enough cars have been selected the sorting can commence. If however some cards have been omitted or incorrectly labelled, this can be remedied by removing or adding cards.

The user immediately selects "Label groups" as the first few sort criteria are already in mind. Group names of **small**, **medium** and **large** are typed in and a sort criterion of **size**. These groups are then shown on the display. The user moves the cards around using the mouse, until each card (or if it is too large the left hand edge of each card) is in the right group. Once done, the "Finish sort" button is

pressed, CATO reads the sort and updates its knowledge base. This continues with the user making sorts based on **price**, **speed**, **country** and **style**. The sort **style** has groups which are disjunctions of styles, such as **hatch/saloon**.

Then the user has run out of obvious sorts and begins to play around with the cards a little. He realises, following on from browsing the previous sort criteria, that not all the cars are in works rally teams, and so makes a sort based on rallying.

Then an analysis of the knowledge is performed and it is shown that **fiesta** and **robin** are not discriminated. This immediately suggests the obvious difference as the next criterion, namely **make**. So the expert makes a sort based on this criterion. The screen display so far is shown in figure 2.

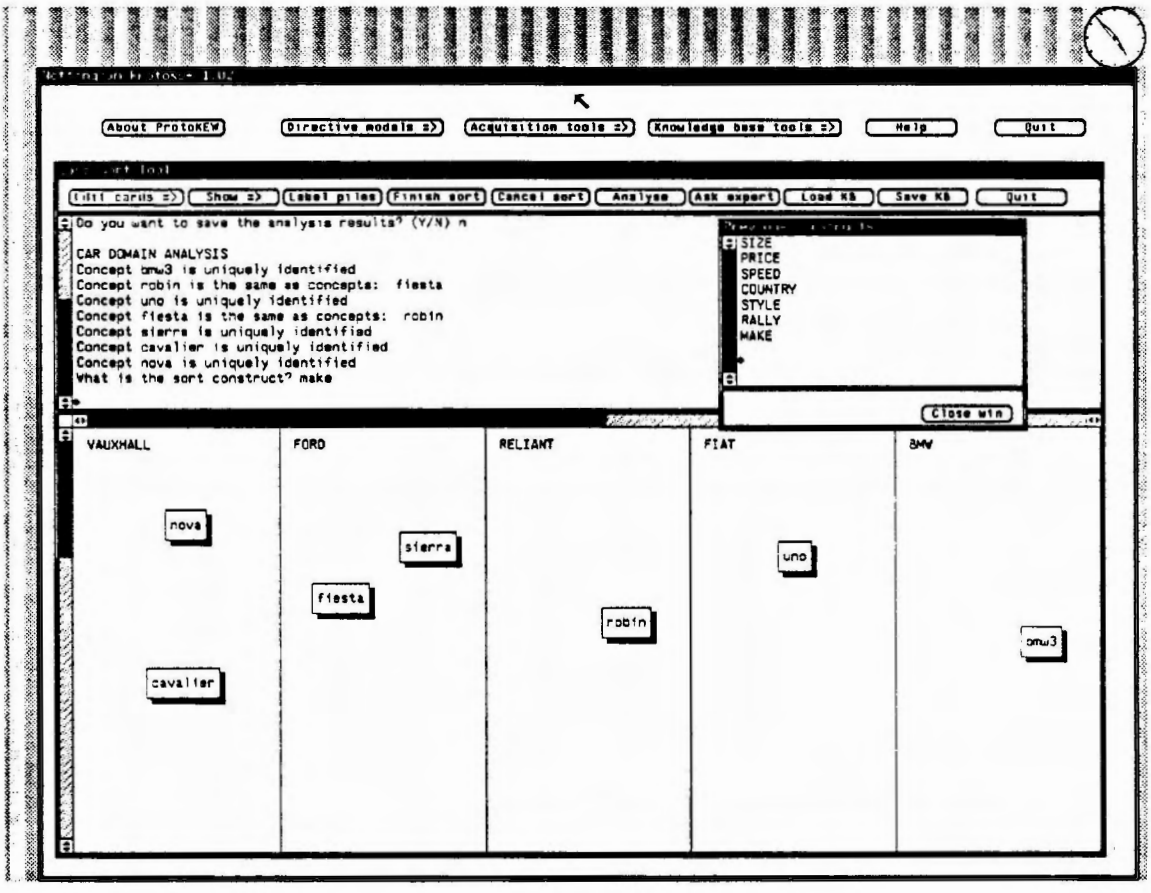


Figure 2: The screen display after seven sorts

Further playing with the cards causes the expert to concentrate on one particular card, **robin**. To see what groups this card has been in he chooses the "Ask expert" button and asks for all the groups for the concept **robin**. The reply omits the crucial fact that the **robin** has only three wheels, and so number of wheels becomes the criterion for the next sort.

At this point a further analysis is carried out and it is found that all the concepts have been uniquely discriminated. Satisfied with this as a basis for ending the session

the expert selects the "Report" button, displaying the whole matrix for all the sorts so far, and saving a version of the matrix to disk. The matrix is shown in figure 3.

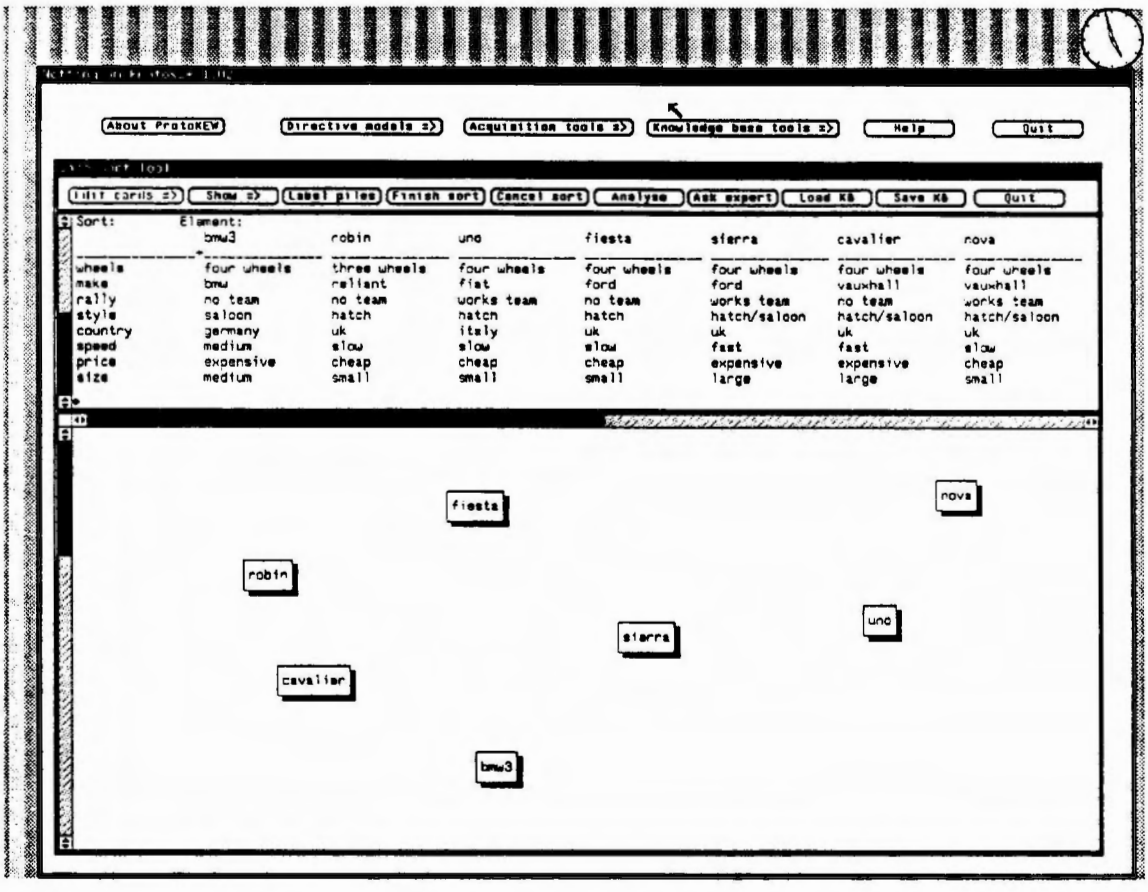


Figure 3: A report of the sorts

When he has exited the tool, he decides to carry out a transformation on the knowledge so that it can be integrated into knowledge about cars found from other knowledge acquisition tools. So he starts up the transformation from CATO to logic. This asks for each concept, whether it represents a class of concepts or a single instance of a concept. The expert answers "class" for all the concepts, and the results of the sorting are transformed into logic. The expert browses the logic propositions that have been generated, but decides not to bother integrating immediately with previous knowledge. The logical propositions are shown in figure 4.

5 Implications for a larger KEW

In section 2 it was pointed out that the sorts of concepts used for card sorting must be at a similar epistemological level. When considering using a card sort within a larger workbench, this choice of concepts becomes all the more important. Care

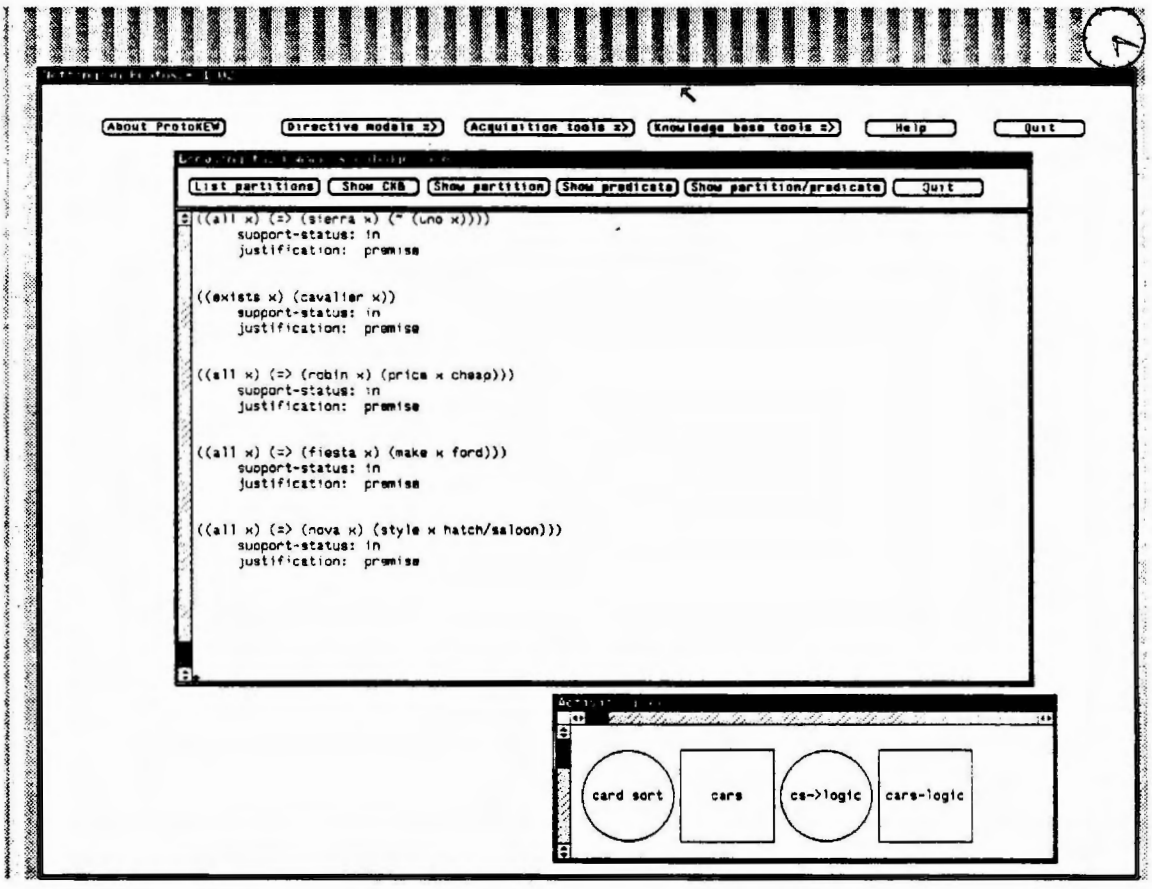


Figure 4: The logical propositions generated from the sorts

would have to be taken to ensure that the concepts used were similar enough, for example by checking any existing relations between them, such as those generated by laddering. Objects represented on the cards would be selected from similar levels in the laddering hierarchy. If no such information was available, it would be necessary to warn the knowledge engineer that large differences could elicit misleading knowledge.

The other significant implication is the consistency of knowledge produced by a card sort tool and by other tools. In CATO this is tackled by transformation into logic, for which well-defined consistency checks exist. It would also be possible to transform the knowledge into frames. However, this gives the problem that is not clear exactly how to check the consistency of different sets of frame-based knowledge. This is especially true if attributes in one representation are frames in another. For this reason it is recommended to use logic for maintaining consistency.

6 Conclusions

The automation of the contrived elicitation technique card sorting has been shown to be successful. The ability of CATO to transform its knowledge into logic immediately begs the question of other acquisition tools doing the same thing. For some time now we have been working on building a prototype workbench [12] which does exactly this. The results of different tools are combined together to form a more complete and thorough knowledge base. A further aim of this prototype is to investigate the extent to which the knowledge engineering process can be automated, or at least supported, by knowledge-based reasoning about the knowledge acquisition process.

7 References

[1] Anjewierden A, Wielemaker J. PCE 3.7 Reference Manual. ESPRIT Project 1098 Technical Report UvA-C1-TR-004, University of Amsterdam, 1990

[2] Bannister D, Fransella, F. Inquiring Man. Penguin, 1980

[3] Burton AM, Shadbolt NR, Rugg G, Hedgecock A. Knowledge elicitation techniques in classification domains. In: Proceedings of ECAI-88, 1988, pp 85-90

[4] Clarke B. Knowledge Acquisition for Real Time Knowledge Based Systems. In: Proceedings of the first European Workshop on Knowledge Acquisition for Knowledge Based Systems, 1987

[5] Grover MD. A Pragmatic Knowledge Acquisition Methodology. In: Proceedings of IJCAI-83, 1983, pp 436-438

[6] Hart A. Knowledge Elicitation for Expert Systems. Kogan Page, London, 1986

[7] Hoffman RR. The Problem of Extracting the Knowledge of Experts from the Perspective of Experimental Psychology. AI Magazine 1987; 8

[8] Johnson PE, Zualkernan I, Garber S. Specification of Expertise. International Journal of Man-Machine Studies 1987; 26

[9] Kelly GA. The Psychology of Personal Constructs, vol 1. Norton, 1955

[10] Major NP, Reichgelt, H. ALTO - An Automated Laddering Tool. In: Wielinga B et al. (eds) Current Trends in Knowledge Acquisition. IOS Press, Amsterdam, 1990

[11] Marty J-C, Shadbolt NR, Wielinga B. The ACKnowledge Project: Improving the Knowledge Acquisition Process. In: Results and Progress of Esprit Projects in 1989, XIII/416/89, 1989

[12] Reichgelt H, Shadbolt NR. Knowledgeable Knowledge Acquisition. In: Steels L, Smith B (eds) Proceedings of AISB-91, Springer-Verlag, 1991

[13] Shaw MLG, Gaines BR. An Interactive Knowledge Elicitation Technique using Personal Construct Technology. In: Kidd AL (ed) Knowledge Acquisition for Expert Systems: A Practical Handbook. Plenum Press, New York, 1987

[14] Weiss S, Kulikowski C. A Practical Guide to Designing Expert Systems. Rowman and Allanheld, Towata New Jersey, 1984

Acquisition of First-Order Knowledge and its application to Air Traffic Control

Christel VRAIN*&**, Yves KODRATOFF*

*Inference and Learning group, LRI, UA 410 CNRS, Bat. 490
Université de Paris Sud, 91405 Orsay Cedex
email: yk@lri.lri.fr

**LIFO, Dept Mathématiques-Informatique, Rue de Chartres
Université d'Orléans, 45067 Orléans Cedex
email: cv@univ-orleans.fr

Abstract

In this paper, we are interested in the way generalization tools can be used to extract expert knowledge. We explain how we perform knowledge intensive similarity-based learning. We introduce first order generalizations and stress their importance to represent accurately the learned knowledge. We present the new problems they raise in knowledge acquisition, namely the proper definition of predicates and the definition of "Skolem functions" by the expert. We see how we have modified the primitive acquisition scheme in an application to Air Traffic Control.

1. Introduction

Everyone agrees that one of the most difficult task when building an Expert System (ES) is the acquisition of the knowledge base. We can wonder whether we could use the progress made in the field of Machine Learning and the developments of new learning tools to make easier this step. It was one of the goal of the project PERSPICACE[1] [4] to study whether ML techniques could be applied to learn the rules that the air traffic controllers use and that cannot be obtained by a direct dialogue with the expert. As it seems easier for an expert to give examples rather than rules of his behavior, we planned to make use of the achievements of the sub-field of ML [6, 7, 8, 10], that deals with the problem of learning a recognition function of a concept by generalization of positive examples of this concept, and avoiding it covering the negative examples.

[1] PERSPICACE : (from the French "Prototype d'Expert en Résolution de Situations, Problème Intéressant le Contrôle Aérien et Conçu à partir d'Exemples") contract C.E.N.A. / L.R.I., october 1986 - october 1988

The knowledge learned can take many different forms. Classically, knowledge is expressed by a set of production rules, as in MYCIN [3, 15]:

if <conditions> then perform <actions>,

where the actions to perform are very often the addition of new facts to the data base. We can find different ways of representing the conditions and the conclusions of the rules. Most often, the representation is based on logical formalisms: propositional or first order logics. Unfortunately, none of them is satisfactory. On one hand, 0th order logic enables very quick inferences but is too much restricted. On the other hand, first order logic has a very high power of expression but matching operations between the knowledge base and the data base may be costly, mainly when using a forward chaining deduction mechanism.

The system we intend to apply enables to learn knowledge expressed as production rules. It uses a technique of generalization that can be described as **knowledge intensive generalization to first order.** In order to make clear what is meant by this, we will have to define carefully what are knowledge poor (or empirical) learning and knowledge intensive learning. We shall show that inductive as well as deductive learning can be knowledge intensive. Within inductive learning, there are also many types of learning. For instance the inductive building of decision trees as performed by ID3-like programs [14, 2] is such a type. It is different from the collection of properties common to the positive examples, and that are not shared by any of the negative examples. We will point out how we are performing knowledge intensive similarity-based learning. We shall introduce first order generalizations and stress their importance to represent accurately the learned knowledge. We present the new problems they raise in knowledge acquisition, namely the proper definition of predicates and the definition of "Skolem functions" (the quotation-marks are necessary and will be explained later on) by the expert. We shall see how we have modified the primitive acquisition scheme in the application to Air Traffic Control (ATC).

2. Knowledge intensive similarity-based learning

2.1. Knowledge intensive versus knowledge poor learning

There are still pending debates in the ML community about the concept of knowledge intensive induction. Many authors link deductive learning [11, 5] with knowledge intensive approaches (which they actually are in existing implementations), and inductive learning with knowledge poor approaches (there exist implementations of induction that are knowledge intensive as well, for instance [10, 6, 7, 17, 18]). One is always using some kind of extra-knowledge, called the background knowledge (**BK**) or the **domain theory**, when performing significant induction. There are actually only two cases, both of them using some kind of BK. In the first case, BK is left implicit. In this case, it is contained in the way the information to induce from is presented. The second case states explicitly this BK. Let us illustrate our point with generalization which is a sub-case of induction.

Technically, inductive generalization is the process which "reverses" the (deductive) generalization inference rule, [$\forall$ x p(x)] => p(A), where A is a constant. Instead of

deducing the instances from the general concept p(x), we induce p(x) from some of its instances $p(A_1)$, $p(A_2)$ ... $p(A_i)$. In general, one is given a set of instances, $p^1(A_1)$, $p^1(A_2)$,..., $p^j(A_i)$; one does not even know if they are related to the same concept.

- The problem of clustering the instances into groups illustrating one concept in each cluster has been called conceptual clustering [10, 8]. In this paper, we did not attempt to use ML techniques to cluster the examples of conflict resolution provided by the expert.
- Once an automatic or manual clustering has been performed, the problem of finding an intentional description [1[1]] of the cluster, viz. describing the instances contained in the same cluster by their common properties is exactly the one of finding a common generalization for a given set of examples. There can be knowledge poor or knowledge intensive generalizations. Knowledge poor generalization, also called similarity-based learning, uses the examples in the form they are given (which actually does contain a large amount of information, but this information is implicit), proceeds by collecting the features that are shared by all the positive examples of the concept. It happens also that negative examples are used, in that case the constraint is that no negative examples share the features of the generalization of the examples.

Let us consider for instance the concept of "good driver". Intuitively, we want to learn a recognition function of the concept, which enable us to determine whether a given driver is a good driver or not. For instance, let us suppose that we get the following descriptions. "John is a man, he is forty years old, he is married and he has never had a traffic accident since he got his driving licence and he is considered by his insurance company as a good driver"; "Mary is also a good driver, she is a woman, she is fifty years old, she is married and she has never had a road accident"; "Peter is twenty years old, he is a man and is unmarried, he was responsible for a very serious accident last year and so he is considered as a bad driver." We can sum up the three events by the following table:

	sex	age	married	accident	driver
John	male	40	yes	no	good
Mary	female	50	yes	no	good
Peter	male	20	no	yes	bad

Table 1

A knowledge poor generalization technique will collect the features common to the examples. Therefore, it would learn here that a good driver is married, and had no traffic accidents since driving licence delivery. This method does not use explicit BK. It uses a large amount of implicit BK in order to transform the given sentences into the table above. For instance, it is not altogether clear that only three features, namely "sex", "age", and "marital status" are contained in the sentences describing the cases.

On the other hand, a knowledge intensive generalization method explicitly uses knowledge about the domain in order to refine the recognition function to learn. For instance, in the previous example, "sex" is an attribute which can have two values "male"

[1] Arnaud and Nicole, known in French as *la Logique de Port Royal* (Port-Royal Logics), actually entitled: *La logique ou l'art de penser* ("Logics or the art of thinking") was first published in 1662.

or "female", and we know that "sex_male" and "sex_female" generalize to "any_sex". Therefore, we could generalize the sexes of our good drivers, and add that they have "any_sex" in the generalization. Many such possible "generalization rules" [10] can be used to improve a generalization. In general, we shall say that this knowledge can be expressed as sets of theorems relative to the domain. Suppose that we know that a person who is between thirty and sixty years old is middle-aged, that is to say that we have theorems such as [30 ≤ age ≤ 60 => middle-aged], [16 ≤ age ≤ 29 => young]. Then, using this extra-knowledge, we can rewrite table 1 as follows.

	sex	age (precise)	married	accident	driver	age (broad)
John	male	40	yes	no	good	middle-aged
Mary	female	50	yes	no	good	middle-aged
Peter	male	20	no	yes	bad	young

Table 2

Applying now the same knowledge poor techniques as before we shall learn a more precise recognition function of a good driver, that is, a good driver is married, had no traffic accidents since he got his driving licence, and is middle-aged.
We therefore call knowledge intensive generalization a generalization that takes into account knowledge about the domain, and embodies it into the examples.

2.2. Knowledge acquisition and Learning

Let us suppose that we want to learn expert rules, expressing the behavior of a car driver. An Expert System approach consists in finding a good driver and asking him to describe the actions he performs while driving and how he executes them.
The approach suggested by similarity-based learning, consists in finding a good driver, observing him when driving to get examples of his behavior, determining which actions he has performed and then for each action, generalizing the conditions in which it has been executed. The rules we learn are all the more interesting as we can use relevant background knowledge.

At the beginning of the process, we have to acquire the domain theory, to determine the concepts to learn; for each of them, we need positive and negative examples. As already mentioned, the process of generalization is an inductive one and cannot guarantee the validity of the results. Therefore, we need to validate the learned knowledge. We shall not take this question up in this paper.

It can be summed up by the scheme of figure 1.

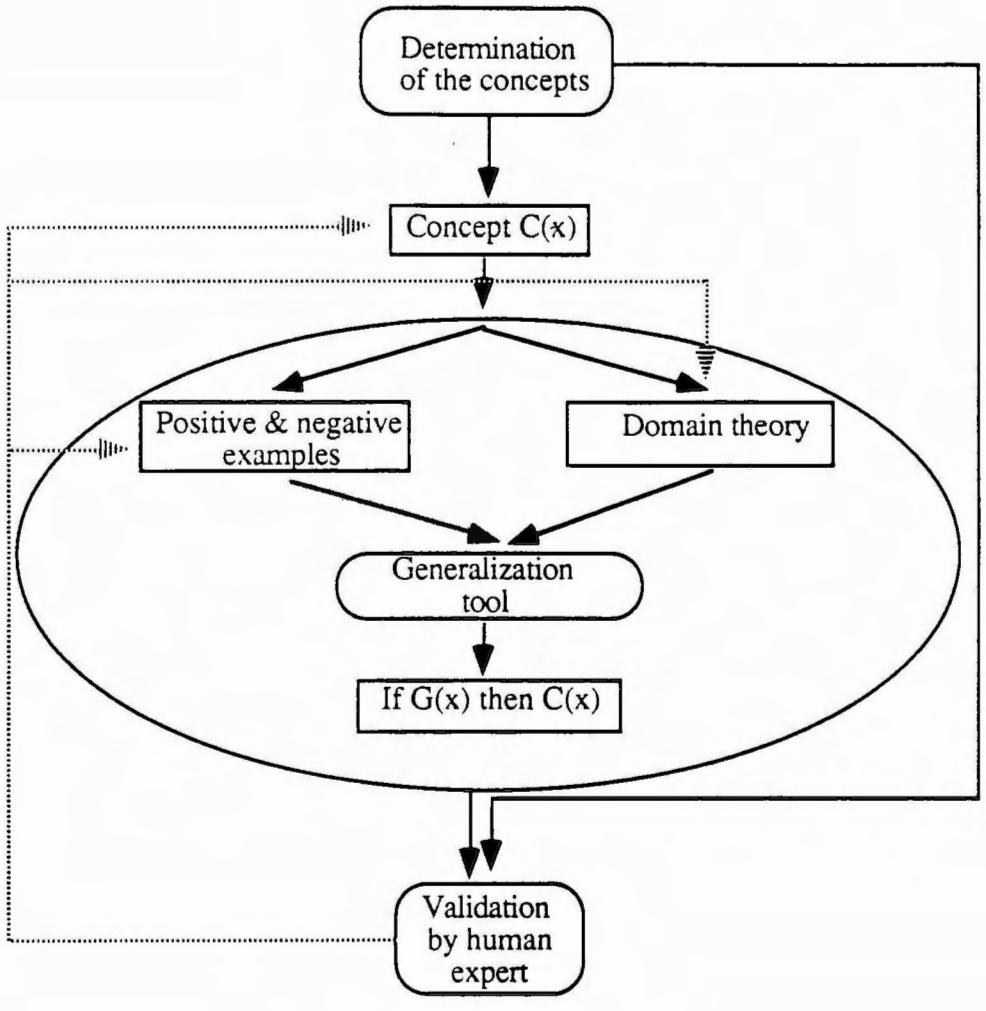


Figure 1

In fact, the application of ML techniques is not as straightforward as suggested by this scheme. Collecting the examples is an easy task for an expert - in the application to ATC, we got in a month roughly hundred examples - but elaborating the domain knowledge is a difficult one.

Beside the dialogue with the expert, we have to find a language of representation adapted to the domain and to the generalization tool and to translate the domain theory, the positive and the negative examples into this language. Most of this paper is devoted to this problem: we chose a first order logic formalism, this choice was mainly justified by the application domain, and we had to face problems inherent to this kind of representation.

3. First order knowledge versus 0th order knowledge

3.1. Justification of first order logic

We have already mentioned that in Expert Systems there is a tradeoff between the power of expression of first order logic and the difficulty to implement efficient algorithms. We

encounter also this problem in ML: for instance in similarity-based learning first order knowledge enables to represent complex events, involving different objects and their relations, but knowledge intensive generalization tools are difficult to implement and have to face combinatorial problems. Nevertheless, we cannot give up this kind of representation, because some domains can hardly be formalized in a 0th order representation.

3.1.1. Some descriptions have the same complexity in 0th order logic as in 1st order logic

Let us suppose that we have n objects, called O_1, O_2,..., O_n, and that they satisfy the following relations: O_1 is at the left of O_2 which in turn is at the left of O_3, and so on until O_{n-1} which is at the left of O_n. In a first order representation, we introduce a predicate "left", n constants representing each of the objects; we write:
$left(O_1, O_2)$ & $left(O_2, O_3)$ & ... & $left(O_{n-1}, O_n)$
In a 0th order representation, we introduce (n-1) propositions, that we call $left_O_i_O_{i+1}$, for i = 1,..., n-1; we write:
$left_O_1_O_2$ & $left_O_2_O_3$ & ... & $left_O_{n-1}_O_n$.
In each case, we need (n-1) atoms or (n-1) propositions.

3.1.2. Some descriptions are a lot more complex in 0th order than in 1st order

Let us suppose now that we know that the relation "left" is transitive. This knowledge can be expressed in a first order representation by the following theorem:
$\forall x \forall y \forall z$ [left(x,y) & left(y,z) => left(x,z)]
In a 0th order representation, to express this piece of knowledge we have to introduce (n-2)(n-1)/2 new propositions, $left_O_i_O_j$, for j = i+2,..., n and we can introduce a set of theorems:

$left_O_1_O_2$ & $left_O_2_O_3$ => $left_O_1_O_3$

...

$left_O_1_O_{n_1}$ & $left_O_{n_1}_O_n$ => $left_O_1_O_n$
$left_O_2_O_3$ & $left_O_3_O_4$ => $left_O_2_O_4$
$left_O_2_O_4$ & $left_O_4_O_5$ => $left_O_2_O_5$

...

$left_O_2_O_{n-1}$ & $left_O_{n-1}_O_n$ => $left_O_2_O_n$

...

$left_O_{n-2}_O_{n-1}$ & $left_O_{n-1}_O_n$ => $left_On-2_O_n$

Note that this set of theorems is far from being exhaustive. Alternately, we can rewrite the description of the example as follows.

$left_O_1_O_2$ & $left_O_1_O_3$ & ... & $left_O_1_O_n$ &
$left_O_2_O_3$ & $left_O_2_O_4$ & ... & $left_O_2_O_n$ &
... &
$left_O_{n-1}_O_n$

In the case of a first order logic representation, we need (n-1) atomic formulas and a theorem and that in a 0th order representation, we need at least n(n-1)/2 propositions.

Moreover, in this last case, the way to treat knowledge expressing relations between objects is neither natural nor straightforward.

3.1.3. Some descriptions cannot be achieved in 0th order logic.

3.1.3.1 - Relations between variable entities

There are relations between parts of objects such that the parts are not instantiated in advance, we know only some of the properties they must fulfill, and the actual instances must be computed on the fly.

3.1.3.2 - Functions

Another point in favor of a first order logic representation is that it enables the representation of functional expressions. For instance, let us consider the relation between the speed, the distance and the time. We are unable to express it in a 0th logic representation, whereas we can write in first order logic:

$$\forall x \, \forall t \;\; distance(x) \;\&\; time(t) \;\Rightarrow speed(x/t)],$$

i.e., if the distance is x and the time is t then the speed is x/t.

In the application to ATC, we have only a few objects, at least the two aircraft in conflict, but we have many relations between them. The expert will forget some of them; he will represent only the pieces of knowledge that he has in mind at a given time. In effect, it means that we cannot express all the knowledge in the initial representation of the examples, we need theorems that will enable to introduce new pieces of knowledge in the examples. Most of the knowledge of the expert is first order knowledge like for instance the previous relation between speed, time and distance.

3.2. Clauses and their skolemization

First order logic allows to express a large variety of knowledge. Nevertheless, it has a well-known drawback, namely the undecidability of the proof procedure. Since we want to use a theorem prover that is efficient enough to apply it numerous times on the examples, we have to restrict ourselves to the simple case of Horn clauses.

Transforming a theorem into a set of clauses includes a step known as skolemization; it happens because in a clause all variables have to be universally quantified. Therefore, all existentially quantified variables have to be deleted.

Let us exemplify the skolemization problem on the clause representation of the sentence: Everybody makes mistakes.

In first-order logic this reads:

$\forall x \, \exists \, y \; [human(x) \Rightarrow (does(x, y) \;\&\; mistake(y))]$

The variable y is under the scope of the ∀ quantifier that quantifies x; therefore y has to be skolemized as a function of x. We can bring in an arbitrary skolem function and transform our theorem into:
∀x [human(x) => does(x, g(x)) & mistake(g(x))]
where g is a new function symbol. In practice, it is awkward to neglect any information we have about the function "g", which tells us that what is done is a misfit action. An expert would even insist on making explicit the kind of actions that are misfit in the considered field of expertise. Suppose that we are in the context of a school class, and the teacher wants to insist on the fact that lack of attention brings mistakes, then he would attribute these errors to unthoughtfulness, and write, instead of g(x), unthoughtful_action(x)
∀x [human(x) => does(x, unthoughtful_action(x)) & mistake(unthoughtful_action (x))].

To complete the example, let us point out that the expert would also dislike the presence of functions in his predicates, he would then rather give to the theorem the form:
∀x ∀y [(human(x) & unthoughtful_action_pred(x,y)) => (does(x, y) & mistake(y))],
where unthoughtful_action_pred(x,y) represents the graph of the function unthoughtful_action(x) and can be expressed in a logic with equality by:
∀x ∀y unthoughtful_action_pred(x,y) <=> y = unthoughtful_action(x).
This last version is no longer equivalent at all to the general theorem we started from. In theory, this is an abomination. In practice, we want the expert happy with the knowledge he has been transmitting into the system. The skolemization process has forced him to make his thought more precise than it was at the beginning, which is a very positive feature of knowledge acquisition.
The choice of an appropriate replacement for a skolem function is a problem which nobody has envisaged an automatic solution to as far as we know. However, as the following examples clearly show, it is quite all right to convert a sentence expressing a feature into a theorem when its conversion into a clause (or rule) requires a "skolemization" which can only be done well by a domain expert. Techniques for acquiring knowledge to construct Expert Systems must not fail to get the expert to point out the right supplementary information that allows to avoid the problem of skolemization.

3.3. How to represent knowledge when using predicates

In most applications, an expert describes a situation he has to face by using attributes and relations between objects. The attributes represent either global descriptors of the context in which the situation occurs as for instance the weather, or descriptors of the objects appearing in the situation as for instance, the temperature of the patient, the color of an object, ... In other terms, the description we get is composed of:
- some pairs (attribute, value) characterizing the environment as for instance (weather, fine),
- some triplets (attribute, object, value) as for instance (temperature Bob high),

- Some relations (relation, object1, object2, . . .) which may be oriented - in this case, the order of the objects occurring in the list is important - or not. Most often, the relations are binary ones.

We see at least four ways of transforming a pair *(attribute, value)* into a first order representation but the list is not exhaustive. Let us consider first the case of global descriptors, represented by pairs *(attribute, value)*, as for instance the pair (weather, fine) which means that for the given situation, the weather is fine.

1st representation: *attribute(value)*, *attribute* becomes a predicate and *value* becomes a constant,
This representation is the closest from the attribute-value one and therefore is the first to come in mind. Our previous example becomes: wheather(fine).

2nd representation: *value(attribute)*, *value* becomes a predicate and *attribute* becomes a constant,
We shall write fine(weather): "weather" becomes a constant and "fine" a predicate. Intuitively, "weather" is interpreted as an object which satisfies the property of being fine. This representation seems closer to our common sense than the previous one.

3rd representation: *pred(attribute, value)*, *pred* is a new symbol of predicate and *attribute* and *value* are constants,
Another kind of representation is to "type" the pair (attribute, value) and then to write pred(attribute, value) where pred is a new symbol of predicate. For instance, we could write that the pair (weather, fine) expresses the weather forecast by the representation: forecast(weather, fine).

4th representation: *attribute(C) & value(C)*, *attribute* and *value* become predicates and C is a new symbol of constants
This representation is close to an object-oriented one. For instance, if we consider our last example, we express that there is an object, called for instance W, which satisfies the two following properties: W represents the weather of the given situation and W is fine: weather (W) & fine(W). The words "fine" and "weather" become predicates and we introduce a new constant W. The idea of introducing a fictitious object is not new. It is used to transform facts written in first order logic into structured object representations [12].

We can apply these four representations to descriptors of objects, that is to say to triplets (attribute, object, value), as for instance the triplet (color_eyes, John, brown). We get:
- 1st representation: *attribute(object, value)* as for instance color_eyes(John, brown),
- 2nd representation: *value(object, attribute)* as for instance brown(John, color_eyes),
- 3rd representation: *pred(object, attribute, value)*, as for instance physique(John, color_eyes, brown),
- 4th representation: *attribute(object, C) & value(C)* as for instance color_eyes (John,C) & brown(C) where *C* is a new symbol of constant.

The first order logic is well suited to express relations between objects, as (relation, object1, object2 ...): *relation* becomes a predicate and *object1*, *object2*, ... become constants and we get *relation(object1, object2,...)*.

4. Application to ATC

The application to a real domain of the knowledge acquisition scheme proposed in the section 2.2. is not as straightforward as suggested by this scheme. We used a generalization tool called OGUST [17,18]. We thought that the main problem would be the use of first order knowledge. In fact the generalization tool we used was well adapted, although it had to face combinatorial problems. The main problems came from the extraction of the domain theory and, surprisingly, from the definition of the predicates, i.e., the expert showed an unexpected propensity to provide us wrong definitions of his own vocabulary.
The acquisition scheme we applied can be summed up by figure 2. We can notice that in comparison with the first figure, we have added three steps of translation: transformation of the raw positive and negative examples and of the theory into a first order representation, translation of the learned rules so that the expert can understand it.

4.1. Transformation of the raw examples

The translation of the raw examples was decomposed into three steps. In real situations, the expert draws a blueprint representing the conflict situation and adds to it necessary information not represented on the drawing. Later, he transcribes them into answers to a questionnaire that can be viewed as a list of attributes with their possible values. The attributes describe either the global environment or descriptors of each aircraft in conflict. In a third step, the obtained examples are automatically translated in a first order representation. We have presented in section 3.3. four ways of translating them; we choose the fourth one that we call an "object" first order representation for the following reasons:

4.1.1 - The use of the 1st or of the 3rd representation of section 3.3 would lead our generalization tool (called OGUST) to learn absurd recognition functions, when two different attributes share the same value.
For example, let us suppose that according to the first representation given in section 3.3; we represent the case when the present work load of the controller and his foreseen work load are high by *present_work (high) & foreseen_work (high)*.
OGUST is not able to make a difference between the *high* of the present work load and the one of the foreseen work load. They represent for OGUST the same object and they are generalized by the same variable, yielding *present_work (vg1) & foreseen_work (vg1)*. The variable *vg1* represents an object which is together the present work load and the foreseen work load. Of course, such a representation is nonsense since it merges into a unique variable two different objects.

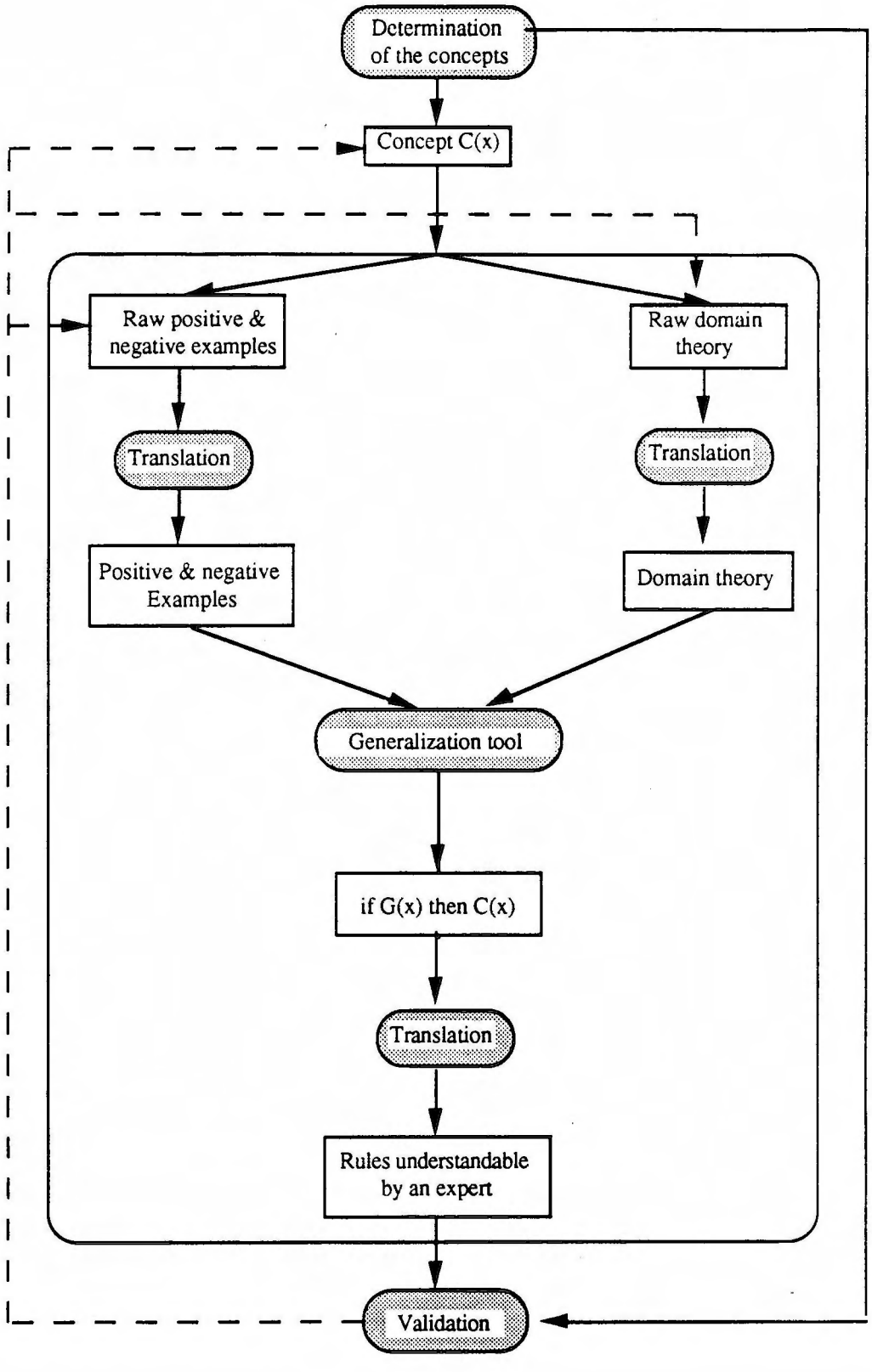


Figure 2

We can give the same argument against the 3rd representation. The previous example would be written for instance, *work (present_work, high) & work(foreseen_work,*

high), and the generalization of the constant *high* into the variable *vg1* would lead to *work (present_work, vg1) & work(foreseen_work, vg1)*.

This problem does not appear with the 2nd and the 4th representations. With the 2nd representation, the example would be written as *high (present_work) & high (foreseen_work)*, and the two distinct constants *present_work* and *foreseen_work* would be generalized into two distinct variables *high (vg1) & high(vg2) [(= vg1 present_work) (= vg2 foreseen_work)]*.
With the 4th representation, the example becomes *present_work (present_work_1) & high (present_work_1) & foreseen_work (foreseen_work_1) & high (foreseen_work_1)*.

4.1.2 - The 2nd and the 3rd representations could prevent OGUST from using BK correctly, thus leading to over-generalizations.

For example, suppose that in E1 the foreseen work load is high, and that in E2 the present work load is high. The 2nd representation gives

E1: high(foreseen_work).
E2: high(present_work).

Suppose also that the attributes *present_work* and *foreseen_work* can be generalized into the attribute *amount_work*..
OGUST uses BK on the predicates, but does not on the constants. Therefore, it generalizes the two constants *foreseen_work* and *present_work* into the variable of generalization *vg1* and learns

G: high(vg1).

It does not use the previous piece of knowledge and looses the information that *vg1* is an amount of work.
The same argument can be applied to the 3rd representation.

This problem arises neither with the 1st representation, nor with the 4th one. Let us consider for instance, the 1st one. The examples are written

E1: foreseen_work(high)
E2: present_work(high),

The constants *high* are replaced in each example, by a variable of generalization, called vg1.

E1: foreseen_work(vg1) [(= vg1 high)]
E2: present_work(vg1) [(= vg1 high)]

OGUST uses its BK to suppress the discriminating occurrences of the variable *vg1* in the examples and we obtain the generalization

G: amount_work(vg1) [(= vg1 high)]

The only representation that solves these two problems is the 4th one, this is why we have been using it.

4.2. Translation of the domain theory

Let us consider the problem of translating the theory in a first order representation and let us illustrate it by the following piece of background knowledge: if a given plane has a steady profile when entering and exiting the sector and in the sector, then it is on cruise. According to the choices we made - the basic vocabulary and the "object" first order representation - we can translate it by the following theorem:

```
[ profile_enter(*x *prof_ent)      &   steady(*prof_ent) &
  profile_in (*x *prof_in)         &   steady (*prof_in) &
  profile_out (*x *prof_out)       &   steady (*prof_out) ]
=>  ∃*fl [ flight(*x *fl) & cruising(*fl) ]
```

which means that under some conditions there exists an object, called *fl that represents the flight and that is cruising. This theorem does not express exactly our knowledge, since it does not express that the object, called *fl, which represents the flight of *x is unique.

According to our generalization tool theorems must be expressed as Horn's clauses, i.e., the conclusion of a theorem must be a single atom and we do not allow existential variables. We replace the previous theorem by the two following ones.

```
[ profile_enter(*x *prof_ent)      &   steady(*prof_ent) &
profile_in(*x *prof_in)            &   steady(*prof_in) &
profile_out *x *prof_out)          &   steady(*prof_out) ]
=>  flight(*x f(*x *prof_ent *prof_in *prof_out))
```

and

```
[ profile_enter (*x *prof_ent)     &   steady(*prof_ent) &
profile_in(*x *prof_in)            &   steady(*prof_in) &
profile_out(*x *prof_out)          &   steady(*prof_out)
flight(*x *fl)]
=>  cruising(*fl)
```

Let us notice that this transformation does not preserve the semantic equivalence between the first theorem and the conjunction of these two new theorems. In this particular case, the two new theorems are right, because the object *fl is unique.

OGUST does not deal with functions and the application of a theorem on an example could therefore introduce a functional term. To prevent this, we apply such theorems by introducing in the example a new constant which represents the value of the function for this example. We lose the information that this constant is a function of other ones.

4.3. Rewriting the generalization

In order to show the learned rules to the expert, we have to translate them in a representation more familiar to him. The module which implements this step reverses the process described in section 4.1.

5. Conclusion

Most people believe that to apply knowledge intensive generalization techniques in order to learn concepts we have only to determine positive and negative examples of the concept, background knowledge and to apply a generalization tool. We have shown in this paper that the process is not as straightforward as it seems.
An important task is the choice of the formalism used to represent knowledge. In most cases, it determines the success of the learning process. The choice of the primitives of the language is too much domain dependent, so that we have no hope of making this step automatic and we can only get them by means of a dialogue with the expert. But, the field of knowledge acquisition could certainly provide us with technics of dialogue that will make easier such an interaction with an expert. This task also includes the choice of the kind of formalism, as for instance, first order formalism versus attribute_value representation. Everyone agrees that first order logic has a greater power of expression than an attribute_value representation but tries to avoid it because the processes that use it lack most often efficiency. Another task is the obtainment of the positive and negative examples and their translation into the formalism we have chosen.
The most difficult problem is certainly the obtainment of background knowledge. We got it by means of a discussion with the expert but we are conscious that it was far from being satisfactory. When expressing if ... then ... rules, the expert forgot very often some conditions that made the rules false. Moreover, he did not give all the rules and then background knowledge was incomplete. This was truly a problem since the falsity of some pieces of knowledge could lead to an erroneous generalization and the lack of pieces of knowledge could lead to over-general generalizations. The step that should validate the learned rules is then essential.
The generalization tool, the interface to stock the examples into an attribute_value representation and the module that translates the examples into a first order formalism have been implemented in Lelisp, version 15.2.

References

[1] Arnaud, Nicole: "La logique ou l'art de penser," *Froman, Stuttgart, 1965.*

[2] Bratko I., Lavrac N.: "Progress in Machine Learning," *proceedings of the 2nd European Working Session on Learning, Bled, Yugoslavia, May 1987, Sigma Press.*

[3] Buchanan B.G., Shortliffe E.H.: "Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project," *Addison-Wesley Publishing Company, 1984.*

[4] Cannat J.J., Vrain C.: "Machine Learning applied to air traffic control," *Colloque International Homme-Machine et Intelligence Artificielle dans les domaines de l'Aéronautique et de l'Espace, Toulouse, France, 28-30 septembre 1988, p. 265-274.*

[5] DeJong G., Mooney R.: "Explanation Based Learning:An Alternative View", *Machine Learning Journal, vol.1, n°2, 1986, Kluwer Academic Publishers.*

[6] Kodratoff, Y., Ganascia, J. G., Clavieras, B., Bollinger T. , Tecuci G., "Careful generalization for concept learning", *Proceedings ECAI-84, Pisa, 1984, p. 483-492.*

[7] Kodratoff Y., Ganacia J. G.: "Improving the Generalization step in Learning", *Machine Learning : An Artificial Intelligence Approach, Vol.2, Eds : Michalski R.S., Carbonell J.G. and Mitchell T.M., Morgan Kaufmann Publishers, 1986.*

[8] Kodratoff Y., Tecuci G.: "Learning based on conceptual distance", *IEEE Trans. on Pattern Anal. and Mach. Int. 10, 1988, p. 897-909.*

[9] Kodratoff Y.: "Introduction to Machine Learning", *Pitman Publisher, 1989.*

[10] Michalski R.S. : "A Theory and methodology of Inductive Learning", *Artificial Intelligence, Vol.20, No.2, feb. 1983.*

[11] Mitchell T.M., Keller R.M., Kedar-Cabelli S.T.: "Explanation Based Generalization: An Unifying View", *Machine Learning Journal, vol.1, n°1, 1986, p. 47-80, Kluwer Academic Publishers.*

[12] Nilsson N.:"Structured object representation", *chap. 9, Principles of Artificial Intelligence, Tioga Publishing Company, p. 361-414.*

[13] Planchon P., Berrada J. : "ANTICO : an anti-collision system in air traffic control", *Colloque International Homme-Machine et Intelligence Artificielle dans les domaines de l'Aéronautique et de l'Espace, Toulouse, France, 28-30 septembre 1988, p. 331-346.*

[14] Quinlan J.R.: "Learning Efficient Classification Procedures and their Application to Chess End Games", *Machine Learning, an Artificial Intelligence Approach, Michalski R.S., Carbonell J.G., Mitchell T.M., eds. Tioga Publishing Company 1983, chap. 15, p. 463-482.*

[15] Shortliffe E.H. : "Computer-Based Medical Consultations : MYCIN", *Artificial Intelligence Series 2, Elsevier Computer Science Library.*

[16] Stepp R.E., Michalski R.S. : "Learning From Observation : Conceptual Clustering", *Machine Learning, an Artificial Intelligence Approach, Michalski R.S., Carbonell J.G., Mitchell T.M., eds. Tioga Publishing Company 1983, p. 331-363.*

[17] Vrain C.: "Un outil pour la Généralisation Utilisant Systématiquement les Théorèmes : le système OGUST", *Thèse de troisième cycle, L.R.I. Université de Paris-Sud, Orsay, France, 25 Février 1987.*

[18] Vrain C. : "OGUST : A System that Learns Using Domain Properties Expressed as Theorems", *Machine Learning, an Artificial Approach, volume III, eds. : Y. Kodratoff, and R. S. Michalski, Morgan Kaufman Publishers, p. 360-382.*

Fifth European Knowledge Acquisition for Knowledge-Based Systems Workshop

EKAW91

Late and Invited Presentations

ACTE: A Strategic Knowledge Acquisition Method

Jean Charlet

Dept. Intelligence Artificielle et médecine
INSERM U194
91, bd de l'hôpital,
75634 Paris cedex 13
France

Abstract

Over the last years, significant progress has been made in understanding how the knowledge acquisition process can be automated. In current knowledge-based systems knowledge is organized according to the reasoning process. Thus, the task to be realized structures the knowledge acquisition process. We would like to further investigate this idea to see to what extent the task specifies the knowledge acquisition tool.

This paper describes ACTE, a method designed to assist in knowledge acquisition for a diagnostic discrimination task. The ACTE implementation shows how the domain concepts and a few problem-solving strategies structure deeply the knowledge acquisition process and how it is possible to take into account the dynamic nature of these strategies for building – in interaction with the expert – an operative and totally specified knowledge base.

1 Introduction

Knowledge acquisition is now accepted as a modeling activity [BB87]. In current knowledge-based systems knowledge is organized according to the reasoning process. Knowledge is acquired for a specific task, according to the way the task is to occur. In a prototypical approach [Cl86], starting from the task to be realized, a qualitative model whose construction can be described in two steps is built. The first one consists in modeling the domain in order to identify the objects and describe the concepts; the second one consists in modeling the reasoning in order to describe behaviors and mutual relationships between these objects. When this modeling act is done it can be used to guide the knowledge acquisition process. Thus the task to be realized structures the knowledge acquisition process [BG89].

The work described in this paper puts into practice this assertion. It presents ACTE, a method designed to assist in knowledge acquisition for a diagnostic discrimination task. We want to show how the domain concepts and a few problem-solving

strategies structure deeply the knowledge acquisition process and how it is possible to take into account the dynamic nature of these strategies for building — in interaction with the expert — an operative and totally specified knowledge base.

The paper is organized as follows. Section 2 recalls the terminology of knowledge acquisition systems and then positions the ACTE method against knowledge acquisition problems. Section 3 presents the ACTE method and the next its implementation in the LÉZARD system. The last section comments some features of the approach.

2 Knowledge Acquisition and Reasoning

2.1 Terminology

Although there is increasing consensus on the importance of the modeling approach, the knowledge acquisition litterature only begins to have a general terminology. In this paper, I shall use the clearly defined terminology proposed by [Mu89].

In developing a knowledge-base, the system-builder seeks to describe a model of expert behavior. This model attempts to approximate — not reproduce — the problem-solving steps of human reasoning [Cl86]. This model is thus a problem-solving method refering to an abstract solution mechanism. This problem-solving method refers to a set of (problem-solving) strategies which work towards the problem solution.

Researchers in Artificial Intelligence (and other disciplines) have identified a number of problem-solving strategies, domain-independent — like [Cl85] for *heuristic classification* — or domain-independent — like [Sz88] for medical diagnostic. Theses strategies are abstracts mechanisms. For example, Clancey's [Cl85] model of the method of *heuristic classification* includes abstract notions such as (i) *conclusions that the problem solver may select from a pre-enumerated set*, or (ii) *data-abstraction hierarchies that alllow the problem solver to generalize from specific input data.* In the same way in diagnostic decision making, most of the systems are more or less based on the two following strategies [Sz88]: (i) *focus on the highest abstraction levels: the program tries to decide if the most general disease classes are present before deciding on more particular classes* ; or (ii) *unicity of the fault process: the program makes attempts to create a unique, causal chain which explains the observed manifestations.*

From strategies, the system-builder builds a model of specific task which represents terms and relationships for defining the application problem to be solved in a specific domain.

Recently, works (MOLE [Es88], SALT [Ma88]) have allowed development of knowledge acquisition tools enabling generation of a model of specific task from a pre-existing problem-solving method peculiar to a set of tasks. A user of these tools has at his disposal strategies and their organization — a problem solving method. He also has an *intension* of the system behavior. Thanks to this tool and specific knowledge of the domain, he builds a knowledge base for a defined application problem: this base is an *extension* of the problem-solving method [Ad87].

Like MORE [Ka85], MOLE or SALT, the ACTE method fits in with this approach: from an *intension* of the system behavior — the strategies used in medical diagnostic

discrimination — the user builds a knowledge base specific to his application — here, diagnostic of acute abdominal pains.

Since this knowledge-base must be used, it must be efficient. Moreover, it must represent the knowledge the expert wants to express. Therefore, it is necessary to answer to the question: "does the model represent the dynamic strategies it is the *extension* of?".

The response is not easy. Obviously, the knowledge representation language cannot represent exactly the knowledge the expert wants to give the system. This perspective is articulated by [Ne82] in his "knowledge-level".

2.2 Knowledge- vs Symbol-level

This perspective is that knowledge is an abstraction that can be separated from the symbols that are used to represent it. Obviously, knowledge representation is different from knowledge itself. This constatation allows to understand the problems looming up when we build a knowledge-base: this difference between the *knowledge-level* and the *symbol-level* prevents the behavior of the system from being completely expressed in the knowledge representation language of the system. The knowledge corresponding to the behavior of the system can only be observed as an activity. Thus, any knowledge representation language fails to completely represent the behavior of the system.

A way to solve this problem is to implement a process allowing to observe system behavior as an activity. Thus, the knowledge acquisition tool must allow the expert to control representation and utilization of the model of specific task he has built. It must allow to verify relevance and correctness, taking into account the dynamic property of knowledge. In some cases it has to activate the model in order to catch its behavior in function.

Thus, ACTE proposes a knowledge acquisition method that approach the above mentioned properties.

3 The ACTE Method

The ACTE method proposes a way to easily acquire a specific task model, from strategies for medical diagnostic discrimination. It has been applied to acute abdominal pains diagnosis.

A first model is initially provided by the expert using a knowledge representation language. Then ACTE "understands" this task model, on the basis of the problem solving strategies it knows, and produces, in interaction with the expert, dynamic knowledge based on these strategies. Finally, ACTE generates production rules from this elaborated task model, and questions the expert to complete the knowledge acquisition process.

The primitives of the knowledge representation language and the ACTE's procedures for constructing rules are described elsewhere [Ch89]. The focus of this paper is on ACTE's knowledge acquisition strategies.

3.1 Knowledge Representation

The knowledge representation language is the way for the expert to describe a first model of specific task. Thus, It allows to model the domain and describe a part of the proposed System's behavior. This first model is describe in a high level representation language which allows to describe both (i) semantic network and (ii) evocation rules.

The network is made of causal and hierarchical relationships between diseases concepts (in medicine concepts are signs, syndroms and diseases).

The evocation rules are expressed in a propositional, logic-like formalism. In medicine they are used for representing the relationships between signs and diseases or between syndromes and diseases.

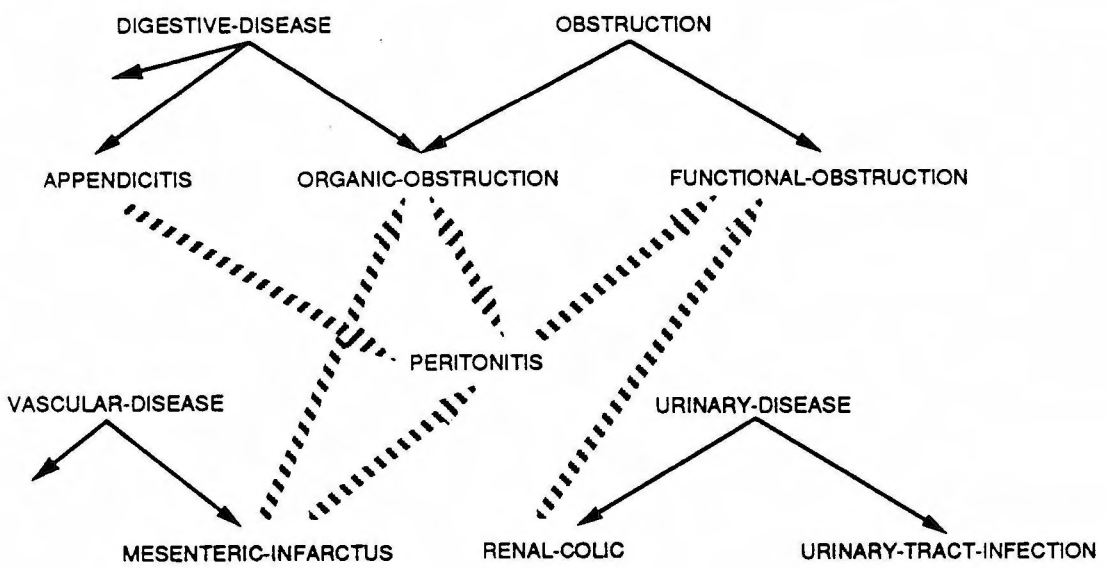


Figure 1: Semantic network representation. Dotted lines represent causal links and Black lines, taxonomic links

3.2 Semantic Network Interpretation

Several diagnostic systems, and particularly medical diagnostic systems [Lo84, Sz88], are based on the hypothesis of "the unique fault process". According to this hypothesis one considers that multiple disorders (in medicine: multiple diseases) may occur simultaneously if and only if they are connected by causal relationships.

This hypothesis makes sense from a medical point of view. It is the first one made by a physician faced with a patient. Exceptions arise with patients whose general states are very deficient, e.g. a very old person, and who suffer from several independent disorders.

ACTE, a knowledge acquisition method in a medical domain, makes use of this hypothesis. According to it, ACTE interprets the semantic network in order to propose an *extension* of this strategy in its domain. Practically, the description of this *extension* consists in *mutual-exclusion* — i.e. two concepts of the domain cannot belong to an unique solution — versus *mutual-compatibility.*

The basic idea of the algorithm is, according to the strategy, "two concepts

without causal relationships are exclusive". Obviously the reality is more complex insofar as the causal relationships may be transitive or not; this leads to ask the expert about transitivity which influence the calculus. The algorithm is optimized for computing this knowledge at the highest level in the hierarchical representation of the domain concepts [CG89].

For example, according to the network of figure 1, we have the following *mutual-exclusions* between diseases :

DIGESTIVE-DISEASE and URINARY-DISEASE are in *mutual exclusion*
VASCULAR-DISEASE and URINARY-DISEASE are in *mutual exclusion*
VASCULAR-DISEASE and APPENDICITIS are in *mutual exclusion*
OBSTRUCTION and URINARY-DISEASE are in *mutual exclusion*
OBSTRUCTION and URINARY-TRACT-INFECTION are in *mutual exclusion*
PERITONITIS and URINARY-DISEASE are in *mutual exclusion*

Moreover, this network interpretation provides a mechanism for detecting potential inconsistencies in the model since the expert can check whether or not the *mutual-exclusions* correspond to his idea.

The semantic network interpretation is then a dynamic simulation of the model of specific task. This interpretation leads to consider the basic inferences performed around the *unicity of the pathophysiological process* strategy. This is a way to implement and check the active behavior due to this strategy.

3.3 Abduction

Abduction proposes a different but direct logical interpretation of causal relationships. It is acknowledged to be useful in medicine [Sz88] and more generally in diagnostic tasks. The principle is as follows: "if I know that Q is true and Q is caused by P, then I can suppose that P is true". The abduction transforms "Q caused by P" into "Q $\Rightarrow$ P".

For example, if ACTE knows that PERITONITIS may be caused by APPENDICITIS, it proposes to the expert the following rule: "PERITONITIS $\Rightarrow$ APPENDICITIS". The implicit purpose consists in asking the expert whether PERITONITIS always has APPENDICITIS as a cause.

There exists other way of interpreting causal relationships. Its proposes to derive the implication "APPENDICITIS $\Rightarrow$ PERITONITIS" from "PERITONITIS may be caused by APPENDICITIS". The implicit purpose consists in asking the expert if APPENDICITIS always causes a PERITONITIS.

3.4 Formal derivation

The rule building mechanism of the ACTE method is formal derivation. It is a rewrite process whose goal is to interpret knowledge. This interpretation is not a problem-solving strategy. It is a way for ACTE to interview domain expert and acquire additional information for building a model as complete as possible.

Let us show how the formal derivation generates questions about knowledge.

How can the piece of knowledge C_1 "A AND B are-the S", which can be written (S $\equiv$ (OR A B)), be used?

The classic way to use this kind of knowledge is propositional calculus. A second, equivalent way is to transform C_1 into the 7 following production rules:

R_1: $((S \wedge (not\ A)) \Rightarrow B)$

R_2: $((S \wedge (not\ B)) \Rightarrow A)$

R_3: $(((not\ A) \wedge (not\ B)) \Rightarrow (not\ S))$

R_4: $A \Rightarrow S$

R_5: $B \Rightarrow S$

R_6: $((not\ S) \Rightarrow (not\ A))$

R_7: $((not\ S) \Rightarrow (not\ B))$

and then to use an inference engine. Our method is close to the latter.

One of the features of the ACTE method is that it deals with uncertainty (obviously, the system for which ACTE allows to describe a knowledge acquisition tool deals with uncertainty). Propositions, for example C_1 shown above, are believed to be true, or are generally true, but are not certain. Strict formal derivation (mapping C_1 to (AND R_1 R_2 R_3 R_4 R_5 R_6 R_7)), as written above, is not possible. The same is true for propositional calculus. In order to illustrate these remarks, let us give a meaning to C_1, A, B, S and R_1:

S signifies: flip a coin three times consecutively.
A signifies: in the three flips, one will observe at least one head and one tail.
B signifies: one observes heads three times.

The piece of knowledge C_1 is "When flipping a coin three times consecutively, one observes at least one head". C_1 can be considered acceptable (it is true seven times out of eight). However, rule R_1: "If one flips a coin three times consecutively, and if one observes three identical results, then one observes three heads," is unacceptable (it is only true one time for two).

The solution we adopt is to have the expert intervene in the formal derivation so as to avoid possible errors and to specify the knowledge. For example: when C_1 is encountered, rule R_1, among other rules, is generated, and the expert is asked about its validity (or its reliability). When it is valid, or sufficiently reliable, it is kept and put in the knowledge base. In this case the expert is also asked about R_1's reliability level. In the opposite case, R_1 is discarded.

This example shows the necessity to interpret knowledge in order to precise it. In a medical domain it is the same: Taxonomy of diseases are necessary to reasoning about pathophysiological process (In general, there can be many different taxonomies orthogonal and tangled). But taxonomies must be specified by the expert, especially the way they are to be used, as we mentioned above.

One might put forward the reason that giving precise knowledge straightaway would be better. But it is generally difficult to acquire and organize knowledge especially when it is dynamic. The best way is to build a model and refine it [WB86]. Formal derivation is an efficient way to do it.

3.5 Summary

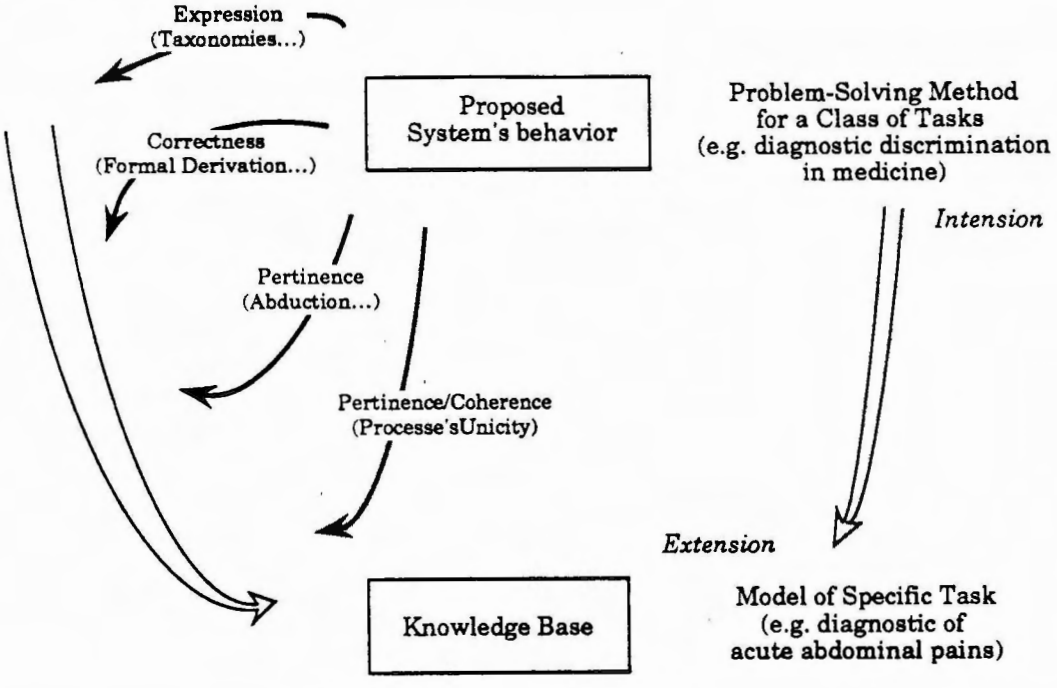


Figure 2: The ACTE method.

We have shown how problem-solving strategies taken into account by ACTE could be seen as mechanisms that help the expert build a model of specific task. In this context, formal derivation is a way to complete, in interaction with the expert, the model built.

The complete knowledge acquisition process performed by ACTE is described in [Ch89]. It includes, among other strategies, some abduction type mechanisms [Sz88] and organizes acquisition steps in a logical way. For example, semantic network acquisition is realized and completed before evocation rules. The latter depending on the former at the level of the diseases.

4 Implementation in the LÉZARD System

The ACTE method is successfully implemented as a knowledge acquisition tool for the LÉZARD system, an expert system currently applied to the diagnosis of acute abdominal pains. The ACTE method allows to build the knowledge-base of the LÉZARD system.

To give an idea of the work performed by the ACTE method for our application, the initial model of specific task contains approximately 135 objects linked by 145 relationships. About 500 questions are proposed to the expert and the final knowledge-base contains about 1200 rules.

5 Concluding remarks

ACTE assists in knowledge-base construction using the strategies described above. Thus, the main feature of this method is that it performs a dynamic simulation of the model of specific task: from a initial model of specific task which fails to

completely represent the dynamic behavior of the system, it leads to considere the problem-solving strategies which control this and build a complete and totally specified knowledge base.

This allows to validate the task model and to reduce the debugging time. Our next step is to take into account new problem-solving strategies for guiding the knowledge acquisition process. In order to do so the medical strategies described by [Cl86, Sz88] are attractive. There are some reasons to believe that ACTE's strategies will apply to discrimination diagnostic tasks in non-medical domains, but this has yet to be demonstrated.

Acknowledgments

Discussions with Jean-Paul Krivine and Bruno Bachimont helped to shape and direct the thoughts above. Thanks to Jacques Bouaud and Pierre Zweigenbaum for her comments on the paper.

References

[Ad87] Addis T.R. (1987). A Framework for Knowledge Elicitation In: *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, reading University, reading.

[BB87] Boose J.H., Bradshaw J. (1987). Expertise Transfer and Complex Problems: using AQUINAS as a Knowledge Acquisition Workbench for Expert Systems. *Int. J. Man-Machine Studies*, 26/1:3–28.

[BG89] Boose J.H., Gaines B.R. (1989). Knowledge Acquisition for Knowledge-Based Systems: Notes on the State-of-the Art. *Machine Learning*, 4/3-4:377–394.

[Ch89] Charlet J. LÉZARD : acquisition des connaissances et gestion de l'incertitude dans un système expert de seconde génération. PhD Thesis, université Paris VI, 1989.

[CG89] Charlet J., Gascuel O. (1989) Knowledge Acquisition by Causal Model and Meta-knowledge. In: J. Boose, B. Gaines, JG. Ganascia (Eds.) *Proceedings of the Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems, Paris*, Express-Tirages , Bourg-La-Reine.

[Cl85] Clancey W.J. (1985) Heuristic Classification. *Artificial Intelligence*, 27/3:289–350.

[Cl86] Clancey W.J. (1986) from GUIDON to NEOMYCIN and HERACLES in Twenty Short Lessons: ORN Final Report 1979-1985. *AI Magazine*.

[Es88] Eshelman L. (1988) MOLE: a Knowledge Acquisition Tool for Cover-and-Differentiate Systems. In: S. Marcus (Ed.) *Automating Knowledge Acquisition for Expert Systems*, Kluwer Academic Publishers.

[Ka85] Kahn G., Nowlan S., McDermott J. (1985). MORE: an Intelligent Knowledge Acquisition Tool In: A. Joshi (Ed.), *Proceedings of the 9th IJCAI, Los Angeles*, M. Kaufmann Inc., Los Altos, California, 581–585.

[Lo84] Long W., Naimi S., Criscitiello M.G., Pauker S.G., Szolovits P. (1984) An Aid to Physiological Reasoning in the Management of Cardiovascular Disease. In: *Proceedings of the Computers in Cardiology Conference*, 1984.

[Ma88] Marcus S. (1988) SALT: a Knowledge Acquisition Tool for Propose-and-Revise Systems. In: S. Marcus (Ed.) *Automating Knowledge Acquisition for Expert Systems*, Kluwer Academic Publishers.

[Mu89] Musen M.A., (1989). Automated Support for Building and Extending Expert Models. *Machine Learning*, 4/3-4:347–375.

[Ne82] Newell A. (1982). The knowledge level. *Artificial Intelligence*, 18/2:87–127.

[WB86] Wielinga B.J., Breuker J.A. (1986), Models of Expertise. In: *Proceedings of the European Conference of Artificial Intelligence.*

[Sz88] Szolovits P., Patil R.S., Schwartz W.B., (1988). Artificial Intelligence in Medical Diagnostic. *Annals of Internal Medicine*, 108:80–87.

A Top-down Approach for Capturing Design Knowledge

Phil Buck, Bryan Clarke, Gareth Lloyd
Logica Cambridge Ltd,
Betjeman House,
104 Hills Rd,
Cambridge CB2 1LQ,
UK.

Abstract. The development of effective computer support for designers needs to be driven by better modelling of the design process, and an effective way of developing such an understanding is by building Artificial Intelligence systems through which this process can be modelled. This methodology has been adopted by the Castlemaine project, a two and a half year industrial and academic collaborative research project investigating the application of AI techniques to the support of the design process. In building a knowledge-based system for supporting drug design we have had to rapidly gain familiarity with a large scientific domain. To achieve this and to specify the requirements for a system, we have employed a set of knowledge acquisition techniques and applied them in a top-down fashion to acquire ever-increasing detail. Techniques have been selected at each level of knowledge abstraction that correspond to the type of knowledge being elicited. Some of the techniques have been applied with the aid of computer-based tools and this has enabled a rapid and efficient capture of knowledge.

Keywords

AUTOMATED DRUG DESIGN; AUTOMATED MOLECULAR DESIGN; CASTLEMAINE; KNOWLEDGE-BASED DESIGN SUPPORT SYSTEM; KNOWLEDGE ACQUISITION TECHNIQUES.

Workshop Category

Methods for capturing design knowledge and requirements.

Introduction

In building a knowledge-based system for supporting drug design we have had to rapidly gain familiarity with a large scientific domain, namely drug discovery, and to acquire designer's knowledge for a sub-field of this domain, indirect drug design. To achieve this and to specify the requirements for a system, we have employed a set of knowledge acquisition techniques and applied them in a top-down fashion with the levels of knowledge being abstracted in progressively increasing detail. Techniques have been selected at each level of knowledge abstraction that correspond to the type of knowledge being elicited. Some of the techniques have been applied with the aid of computer-based tools and this has enabled a rapid and efficient capture of knowledge.

With the increase in the number of large knowledge-based systems being developed, the need to gain familiarity in new domains and to elicit knowledge rapidly and efficiently due to limited access to domain experts, there is a demand that structured and methodological approaches to knowledge acquisition are used. In this paper we describe the techniques we have found most useful in design knowledge acquisition and define the aspects of design knowledge capture for which each technique is appropriate.

In this paper we will discuss :-

- The Castlemaine project - the context for the knowledge acquisition
- The domain of indirect drug design - an introduction to the domain
- The techniques of knowledge acquisition used and the utility of the results obtained
- The implementation of the acquired knowledge - a brief statement on the current status of the implemented software for the knowledge-based design support system
- Future directions in domain analysis - how we would like to see the domain anaysis proceed in the future; particularly with regard to applying a methodological approach

The Castlemaine Project

The development of effective computer support for designers needs to be driven by better modelling of the design process, and an effective way of developing such an understanding is by building Artificial Intelligence (AI) systems that model this process. This philosophy has been adopted by the Castlemaine project[1], a two and a half year industrial and academic collaborative research project investigating the application of AI techniques for supporting the design process ([BCL91]). In the first phase of the project, a prototype knowledge-based design support system integrating various knowledge-based system techniques to aid medicinal chemists in their drug design has been constructed. The development of the system has been underpinned by extensive knowledge acquisition work conducted with the medicinal chemists of British Bio-technology (one of the four Castlemaine collaborators).

The Domain : Indirect Drug Design

Current approaches to drug discovery can be summarised in the diagram shown in Fig. 1.

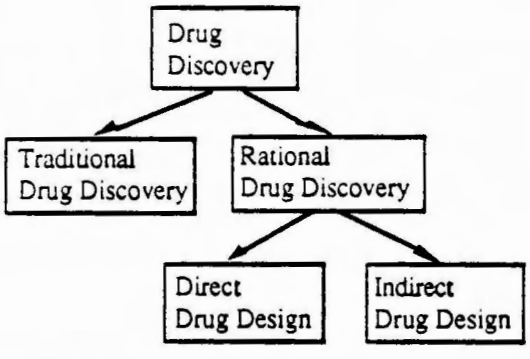


Figure 1 Hierarchy of drug discovery approaches

Traditional drug discovery has relied upon the highly serendipitious method of screening a large set of existing candidate substances for ones with the desired physiological effect. As pharmacological knowledge has increased, so the typical candidate set has grown to contain many thousands of potential molecules. The screening method, though successful in the past, is now increasingly seen as wasteful of research resources and also unlikely to produce the ideal drug.

Increasingly, the approach is to design the drug at the atomic level, based upon an understanding of the relationship between drug structure and drug activity. This approach can be termed *rational drug discovery* and is based upon the precept that the

---

[1] Castlemaine is one of the industrial and academic collaborative projects set up under the Department of Trade and Industry's Information Engineering Directorate initiative, the UK government directed information technology research programme which followed on from the Alvey programme.

pharmacological activity of a drug is a direct consequence of its binding to the "target" receptor molecule. The requirements of the receptor's binding site, the part to which the drug specifically attaches, need to be modelled with a view to defining a drug molecule which can meet these requirements. When the nature of the binding site is known, the task of drug design involves the production of a complementary molecule.

If the molecular structure and geometry of the receptor is known, for example through X-ray crystallography, then the drug designer knows the receptor's requirements exactly and can use the *direct* approach in designing drugs for that site. Unfortunately, the geometry of most pharmacologically interesting receptors has not been characterised at an atomic level and the process of obtaining the molecular structure for a single receptor type may take many years.

More often, the designer has to take the *indirect* approach to drug design. The nature and size of a receptor binding-site can often only be inferred from the drugs that the receptor most readily accepts. This approach is analogous to attempting to infer the inside configuration of a lock from an examination of the keys which best fit it. Most molecules are flexible and it is not known which three-dimensional configuration any given molecule will take. This means that indirect approaches cannot directly model the geometry of the receptor. Obviously, the binding-site model produced from these techniques can only be an incomplete approximation.

The indirect drug-design approach models the "pharmacophore", an arrangement of chemical properties which are thought to be necessary for binding specifically to a particular receptor type. This model is derived from analysis of a set of drugs, all of which produce the desired biological activity to some degree and which are assumed to bind to the same receptor. This model is used to guide the design of new chemical molecules which target the same receptor type. The drug designed must be specific in that it binds to the target receptor but not to any other type of receptor: over-generalisation produces unwanted side-effects in the recipient of the drug. In indirect drug design, the structure of the receptor and its binding-site can be entirely unknown or perhaps only partially postulated in terms of important features; the effort here is concentrated on elucidating the features and functional groups of the drugs which contribute to high activity. Analyses of structure activity relationships (SAR) identify the salient structural features of the compounds contributing to the pharmacophore model, ranking these structures in terms of their importance for activity.

The Castlemaine project has adopted *indirect* rational drug design as the sub-domain of molecular design to be focussed upon for the preliminary prototype because indirect drug design relies on substantial amounts of expert knowledge and experiential techniques more easily captured using AI techniques than those in conventional computing, and more than 95% of the drug design work carried out by pharmaceutical companies is indirect drug design.

Techniques for Drug Design Knowledge Acquisition

A top-down approach has been followed in the domain analysis for the preliminary prototype. As the knowledge engineers have become more familiar with the domain of drug design and the objectives for the preliminary prototype have been defined by the project team as a whole, then more focussed knowledge acquisition techniques have been progressively applied.

The techniques used for the preliminary prototype of Castlemaine fall into four main categories and were pursued in the following order :-

- case studies
- task analysis
- concept structuring
- protocol analysis

The use of these techniques and their contribution to the knowledge acquisition and representation is described.

Case Studies

Case studies were conducted very early in the course of the knowledge elicitation conducted for the domain of drug design. These sessions were mainly conducted in an unstructured fashion; the expert related the historical course of development of an actual past drug design project and the elicitors asked questions to clarify points or to elicit more detail on points of interest. This process provided a familiarisation for the knowledge engineers of the concepts, concept relationships and problem-solving strategies involved in the domain and demonstrated the difference between the two main categories of *direct* and *indirect* drug design. This clarification of the drug design domain supports the taxonomy depicted in Figure 1. The overview of the drug design domain thus gained assisted the later decision by the Castlemaine consortium to focus effort on supporting *indirect* drug design. From the case studies, a set of domain concepts was compiled and an insight into some of the tasks that are performed was gained.

Task Analysis

The first formal task analysis that was conducted with the drug design experts at British Bio-technology was aimed at providing an analysis of the top-level tasks that take place in a drug design project. This included an analysis of the company procedures that are normally followed before the formal instigation of a drug design project and then analyses of the three main approaches used in rational drug design (two indirect approaches and one direct approach). The session was conducted by the drawing of dataflow diagrams on flip-charts during the session.

Subsequent task analyses were aimed at providing a focussed and structured analysis of the lower-level tasks that are involved in the indirect drug design approach

that had been chosen for support in the Castlemaine preliminary prototype. These knowledge elicitation sessions were structured by using the dataflow diagram drawing tool MacCadd™ ([Mac89] on a Macintosh computer. The format of the sessions was for an elicitor and an expert to jointly construct diagrams directly onto the computer screen using the keyboard and mouse controls provided. The MacCadd facilities allow the rapid development and modification of dataflow diagrams and also the ability to maintain a hierarchy of diagrams and sub-diagrams though the "zooming in" facility (a diagram box can be selected to expand and display an underlying dataflow diagram). This interactive technique allowed the expert to immediately see his knowledge displayed in a structured form and to review and modify it during the session. Using this direct input technique, the bias of the elicitor's interpretation can be minimised to a considerable degree. The use of MacCadd in task analysis was found to be very effective - the highly scientific nature of the domain and the concomitant sophistication of the experts to be able to work with symbolic reasoning was a major factor in this success.

Figure 2 shows one of thirteen MacCadd diagrams produced in the task analysis of indirect drug design. This diagram corresponds to the topmost level tasks that were identified. The symbols with rounded ends represent tasks to be performed, rectangles represent goals to be achieved and the computer VDU-like symbols represent tasks that were considered suitable for computer support. Symbols with a greyed inside border indicate that they can be "zoomed in" to display an underlying dataflow diagram.

™ Trademark of Logica UK Limited

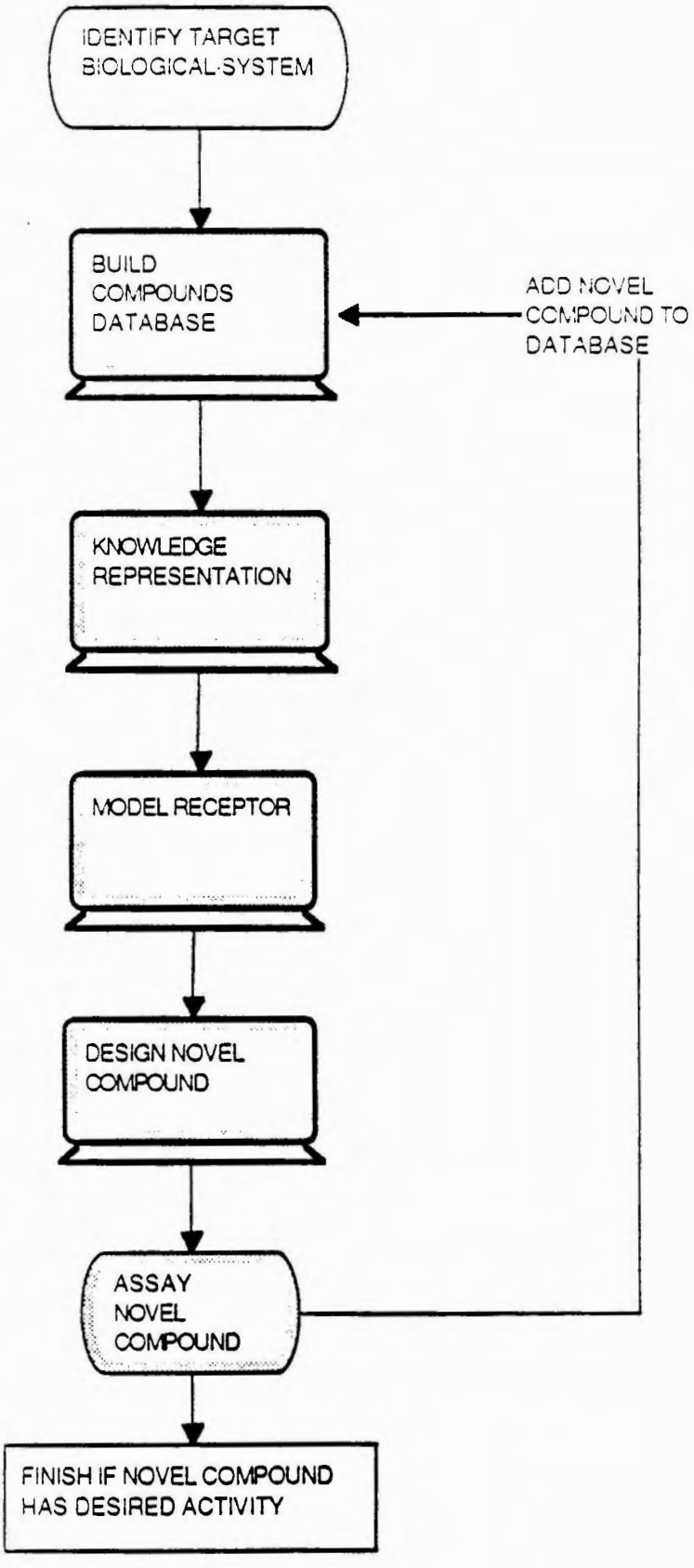

IDENTIFY TARGET BIOLOGICAL SYSTEM
BUILD COMPOUNDS DATABASE
ADD NOVEL COMPOUND TO DATABASE
KNOWLEDGE REPRESENTATION
MODEL RECEPTOR
DESIGN NOVEL COMPOUND
ASSAY NOVEL COMPOUND
FINISH IF NOVEL COMPOUND HAS DESIRED ACTIVITY

Figure 2. Top-level MacCadd diagram showing the top-level loop of drug design

The set of task analysis MacCadd diagrams were reviewed at project technical meetings and the consortium identified four sub-tasks within indirect drug design that appeared appropriate for support within the preliminary prototype system :-

- *Analyse Compound*. Compounds, are partitioned into their smallest, pharmacologically meaningful building-blocks, Molecular Components. These are then assembled into larger meaningful blocks, Molecular Fragments.

- *Compare Compounds*. Compounds are compared against each other to pick out Molecular Fragments that fall into patterns across the compounds. Properties of the identified patterns are described in terms of Features. These are assembled together into a representation termed the "pharmacophore". In Fig. 2, these pharmacophore features are depicted as alternate shadings.

- *SAR Analysis*. Structure-Activity Relationship (SAR) data is elucidated, that is, the relationship between the activity of a compound and its structure is described.

- *Produce New Compound*. The pharmacophore patterns and SAR data is used to suggest modifications that can be made to an existing drug to design a novel drug with improved activity.

Descriptions for these identified sub-tasks formed the bases for the Functional Specification and the Design Specifications for the preliminary prototype.

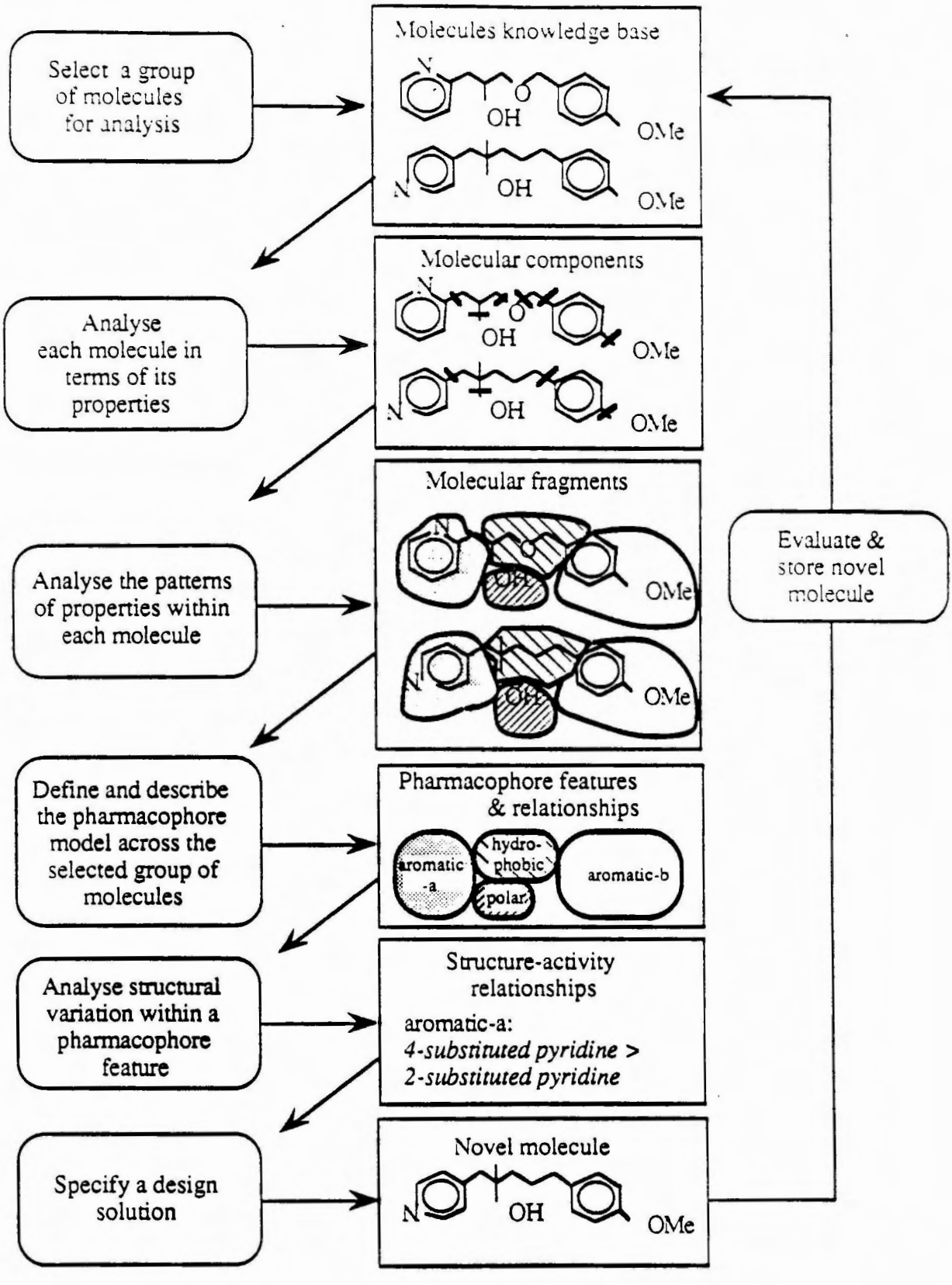

Select a group of molecules for analysis
Molecules knowledge base
Analyse each molecule in terms of its properties
Molecular components
Analyse the patterns of properties within each molecule
Molecular fragments
Define and describe the pharmacophore model across the selected group of molecules
Pharmacophore features & relationships
aromatic -a
hydro-phobic
polar
aromatic-b
Analyse structural variation within a pharmacophore feature
Structure-activity relationships
aromatic-a:
4-substituted pyridine >
2-substituted pyridine
Specify a design solution
Novel molecule
Evaluate & store novel molecule
OH
OMe

Figure 3. The Castlemaine model of indirect drug design

Figure 3 represents the main subtasks being supported (rounded boxes) and the representations which they operate on (rectangular boxes). In this diagram, physical structures are represented as molecular diagrams and property groupings are represented as shaded areas. The top-level feedback loop is given for simplicity but there are, of course, intermediate feedback loops in which aspects of problem analysis and synthesis are explored.

Concept Structuring

1) Laddered Grid

In the KADS methodology, an integral part of the recording of the static knowledge within the domain layer of the conceptual model is the construction of hierarchical tree structures of the domain concepts. These are parent-child structures that can have relations between the nodes of types "is-a", "part-of" and "causes" (amongst others).

The laddered grid technique is one in which the elicitor constructs a well-defined representation of the domain in collaboration with the expert. This can be drawn as a graphical representation of the domain. On the Castlemaine project, the representation was constructed directly into the *frames system* of the knowledge engineering toolkit (Goldworks II™) being used on the project. The drug design experts of BBL were introduced to the *frames system* of Goldworks II™ by the elicitor. This system can be used to structure the concepts of a domain in a parent-child type of tree structure. Each frame can have attributes (termed "slots" in frames terminology) attached to them and these attributes may be inherited from parent frames to child frames ([Ric88]). Using the set of drug design concepts produced from the earlier task analyses, a domain concept hierarchy (termed a "lattice in Goldworks) was begun on the Goldworks system using an "is-a" relationship between the concept frames (e.g. "enzyme" frame is-a "protein" frame is-a "large-molecule" frame etc). The format of this part of the session was for the elicitor and the expert to jointly construct the lattice directly in Goldworks. The elicitor "drove" the system by means of the keyboard and mouse, whilst the expert provided the information needed. This interactive approach enabled the expert to see the lattice as it was built up (by means of the "lattice browsing" and "frame inspection" facilities of Goldworks) and to request modifications as necessary.

The lattice thus constructed was in the first instance constructed as a part of the *static* knowledge of the domain and not to necessarily play a direct part in the inferencing - this is consistent with the KADS methodology. This large lattice of concepts contains many frames that are not needed for the dynamic inferencing of the

---

™ Trademark of Goldhill Computers Inc.

preliminary prototype system. Therefore, the lattice has subsequently been pruned down to a *working* lattice that will structure the knowledge-base for the preliminary prototype.

One of the two branches descended from CHEMICAL-OBJECT of this pruned lattice is shown in Fig. 6, nb. the branch not shown refers to is that part of the lattice descended from MODEL. It can be seen that five types of chemical object: MOLECULE; ATOM; BOND; MOL-FRAGMENT; and MOL-COMPONENT, will be used for reasoning by the preliminary prototype. The lattice represents simple specialisations and composition with the links *is-a* and *has-parts* respectively. For example, from Fig. 4 it can be seen that DRUG is a MOLECULE, which in turn is a CHEMICAL-OBJECT.

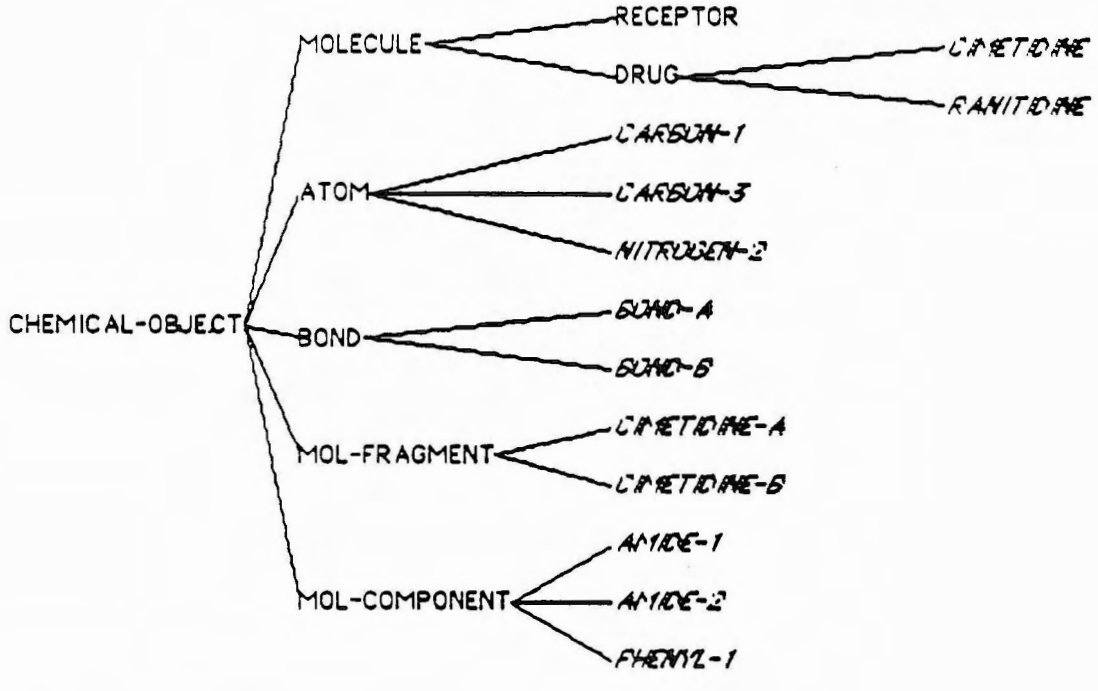


Fig. 4 Part of the Drug Design Lattice

The conceptual map of the domain for the preliminary prototype is now complete and the rules required for the implementation of Support System 1 (Analyse Compound) have been elicited. Analysis for acquiring the knowledge for Support System 2 (Compare Compounds) is essentially complete, though rules are periodically modified as understanding of the domain progresses. The major elements of the knowledge will consist of an analysis of the properties of Molecular Components, Fragments and Features and an elicitation of rules to describe the relationships between these entities and the relationships between the properties of these entities.

2) Repertory Grid

We have used a version of card sort, the Repertory Grid ([WH88]), to make a first pass at ascertaining the properties of molecular components that are salient when comparisons amongst molecular components are made. The aim is to create a "multi-dimensional" map of the elements of the domain i.e. to define the dimensions of difference for the molecular components.

At the sessions, one of the experts wrote down onto a set of 30 small cards the structures and names of the 30 most frequently occurring molecular components. The two experts then took it in turns to select groupings of three cards on the basis of there being two of components which are in some way alike, and a third which is different. The dimensions of difference, or *constructs*, within the groupings were then discussed. Halfway through the exercise, the strategy was changed so that one expert would pick a grouping and then the other expert would have to say what constructs he thought the first expert had in mind in making the selection; this would then be confirmed or denied by the first expert. Generally, there was almost no disagreement between the experts on the assignment of constructs and for this reason the knowledge recorded was not attributed to a particular expert.

One of the groupings and its transcription into a grid of construct values are shown in Figure 5 (the domain elements, molecular components, are shown in bold typeface)

Similar 1	*Similar 2*	*Different*	*Constructs*	*Salient*
phenyl	***naphyl***	***cyclohexyl***		
yes	*yes*	*no*	*aromatic*	√
yes	*yes*	*no*	*planar*	√
no	*no*	*no*	*polar*	
			H-bonding	
smaller	*bigger*	*middle*	*size*	
similar	*similar*	*different*	*conformation*	√
			halogen	

Figure 5. Two of the molecular component groupings

The exercise helped to define the minimal set of properties that a Molecular Component should have and has given both the knowledge engineers and the domain experts a feel for the sort of information that will be required to make a complete molecular component knowledge-base. However, the set of domain elements (the Components) were analysed in a random, non-exhaustive fashion - the experts naturally chose example groupings that they felt were the most interesting and gave less attention to the more mundane groupings. What was then needed was to elicit more of the same kind of knowledge, but to do it in a rigorous and structured fashion.

For this purpose, we have used an "automated attribute acquisition" tool that was developed as part of the Rule Induction programme at Logica Cambridge. The tool guides the user to compare domain elements (Molecular Components) and to enter attributes for the domain and prompts him to fill in missing data. The final result was the compilation of a library of Molecular Components with their properties. For consideration is the option that the results from this tool can then be fed into a rule induction module which will generate classification rules for the domain elements - this looks particularly attractive for the classification of Molecular Fragments. The tool will be evaluated for its role in further knowledge acquisition in the Main Prototype.

Protocol Analysis

The KE technique that was primarily used for Support System 2 (Compare Compounds) analysis was Protocol Analysis. This technique consisted of the expert solving a problem (such as the assignment of Fragments to Features for an example set of compounds) in front of the elicitor, and being asked to "think aloud" during this process. It has been demonstrated, [BSR90], that so-called "contrived" KE techniques such as card sort and laddered grid frequently yield knowledge with greater efficacy than "traditional" techniques such as structured interview and protocol analysis. But Protocol Analysis seemed the most natural way to proceed for this stage of the analysis, particularly since the problem of assigning Fragments to Features is not one that is normally explicitly done by a drug designer and therefore requires a good deal of thought and consideration - the expert frequently found it helpful to "bounce ideas off" the elicitors themselves.

Implementation of the Acquired Knowledge

The knowledge acquired by the above-described techniques has been used in the construction of a preliminary prototype to support indirect drug design. The first three of the identified sub-tasks have been successfully implemented as support systems in this prototype. The fourth sub-task (Produce New Compound) has not been implemented as planned due to the exhaustion of implementation effort for the preliminary prototype in the construction of the first three support systems. Completion of the knowledge acquisition and implementation work for this support system has been postponed to take place in the construction of the main prototype.

However, the first three support systems are able to produce a pharmacophore for a drug design problem - the pharmacophore is the model of the requirements for the new drug. The preliminary prototype has been successfully demonstrated to the Information Engineering Directorate (IED) and is currently undergoing an evaluation phase, but has already been favourably received by medicinal chemists at British Biotechnology.

Future Directions in Domain Analysis

Castlemaine is a project to demonstrate the applicability of design support systems primarily in the domain of drug design. It is to be anticipated that the analyses of these complex domains will result in the acquisition of copious amounts of knowledge; not only existing, explicit knowledge obtained directly from the domain knowledge - but also less accessible, implicit knowledge about the design process underlying those domains (this *design knowledge* will mostly be obtained by elicitation from the domain experts).

For the preliminary prototype analysis, although ideas from the KADS methodology have influenced the analysis, a *formal* knowledge acquisition methodology has not been used on the Castlemaine project. The pattern has been to initially use unstructured interviews to gather overviews of the drug design domain and to impose increasing structure on KE sessions at successive meetings between elicitor and expert (an approach described by [Sha88]). However, as the amount of domain knowledge accumulates and the number of identified inter-relationships between domain concepts grows on an almost exponential basis; the need to structure the knowledge and to define the approach to its acquisition becomes increasingly apparent.

The large amount of effort involved in analysis clearly requires a structure or methodology to be imposed on it for it to be utilized effectively; the Esprit project KADS (see [Hic89]) provides one way of achieving this structuring.The domain modelling approach of KADS provides a means of KA organisation and it is intended that this approach will used as a KA methodology for the Main Prototype. The selection or construction of a suitable *interpretation* model (KADS terminology for an abstracted model of the problem type) for drug design will enable acquired knowledge to be classified into a suitable location in the model and used to its best advantage. This modelling process will also enable areas of the domain that have not been

sufficiently well analysed to be identified as such and have attention focussed upon them. The interpretation model should define what should be known about each task within the model (e.g. its inputs, outputs, methods etc) so that this knowledge can be acquired and recorded. As the KA process continues, the selected interpretation model (this model could have been constructed from a combination of "library" interpretation models) should gradually become instantiated to the *conceptual* model (KADS terminology for an abstracted model of the domain in question) of indirect drug design.

Discussion & Conclusion

The domain for the Castlemaine project, indirect drug design, has been investigated from a starting position of having little formalised knowledge on how the process of designing a new drug is done. This process of starting from "a blank sheet of paper" and progressively expanding the detail and the nature of the domain knowledge is summarised by the diagram in Figure 6.

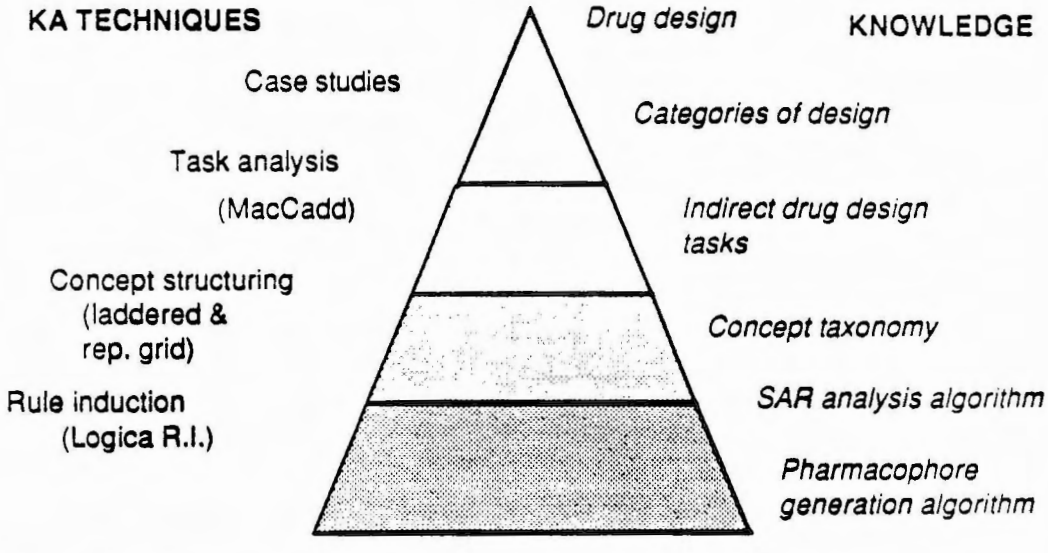


Figure 6. Progress in acquisition of indirect drug design knowledge

A large amount of structural knowledge for the domain (the concept taxonomy) was acquired quickly and efficiently using the "contrived" KA techniques of laddered grid and repertory grid. However, this knowledge was reasonably unambiguous in its nature and thus straightforward to acquire. The non-structural knowledge (particularly the algorithm for pharmacophore generation) is much harder to acquire and requires a large amount of effort for its formulation into algorithms or production rules - it is particularly hard in a design domain. For a classification domain (e.g. medical diagnosis), the expert will usually be attempting to produce a single, correct answer from the evidence presented to him. By contrast, in a synthesis domain, such as indirect drug design, the expert can consider an almost unlimited number of solutions. This lack of constraint on the number of solutions makes it very hard for the

knowledge elicitor to be confident in defining the process by which problem solutions are generated. There is little alternative other than conducting a large number of the traditional type of Knowledge Elicitation sessions; primarily using protocol analysis and structured interview.

Thus, we agree with [BSR90] that contrived KE techniques can yield knowledge with greater efficacy than the traditional KE techniques and where contrived techniques are applicable, then it will frequently be of most cost-effectiveness to apply them. However, it is ultimately a question of "horses for courses" - the right KE technique for the right task.

A highly complex domain, indirect drug design, has been mapped from its highest, most abstracted level (the hierarchy of drug discovery approaches) down to its lowest, most detailed level (the atomic composition and chemical properties of Molecular Components) and appropriate knowledge representations have been produced. System requirements have been elicited by the mapping of the domain task analysis onto a system design. Different knowledge acquisition techniques have been investigated for their utility in the capturing of design knowledge at the different levels of abstraction and the practicality of computer-based tools has been demonstrated.

References

[BCL91] P.Buck, B.Clarke, G.Lloyd, K.Poulter, T.Smithers, M.Xi Tang, N.Tomes, C.Floyd and E.Hodgkin. *The Castlemaine Project: Development of an AI-based design support system.* To appear in forthcoming conference proceedings of "AI in Design '91".

[BSR90] A.M.Burton, N.R.Shadbolt, G.Rugg and A.P.Hedgecock. *The Efficacy of Knowledge Elicitation Techniques: A Comparison Across Domains and Levels of Expertise.* Knowledge Acquisition 2, pp. 167-178, 1990.

[Hic89] F.R.Hickman. *Analysis for Knowledge-Based System: A Practical Guide for the KADS Methodology,* 1989.

[Mac89] Logica UK Ltd. *MacCadd Version 5.0 - A New Generation of CASE for the Apple Macintosh.* January 1989.

[Ric88] E.Rich. *Artificial Intelligence,* 1988.

[Sha88] N.Shadbolt. *Knowledge Elicitation: The Key to Successful ES.* pp. 22-25, Expert Systems User, October 1988.

[WH88] N.Williams and P.Holt. *Expert Systems for Users,* 1988.

The Systematisation of Mathematical Knowledge

Peter Samuels
The Numerical Algorithms Group (NAG) Ltd.,
Jordan Hill,
Oxford OX2 8DR

26 April 1991

Abstract

The systematisation of mathematical knowledge is presented within a proposed framework. Knowledge Acquisition (KA - see [25]) for Mathematics is described with reference to the KASTLE project and the construction of knowledge-based front-end (KBFE) systems. The relationship between Mathematics and mathematicians is discussed.

1 Introduction

During this century, there has been a growing trend in Mathematics to systematise, i.e. to put Mathematics into formal (or pseudo-formal) mathematical systems. Some influences behind this trend have been:

- the more formal approach taken towards Mathematics expressed in the Hilbert Programme and the rise of Mathematical Logic (see [26]);
- the computer and Information Technology revolution and its effect upon Mathematics - see [36] and [29]; and
- the shift in emphasis from viewing Mathematics as a body of knowledge to viewing it as a process - for example several arguments in [49] and the debate over the recent introduction of the National Curriculum in the United Kingdom, with its emphasis on problem-solving rather than factual knowledge (see [17]).

In this paper, a framework for the systematisation of mathematical knowledge is proposed and then applied to different areas of Mathematics. Mathematical knowledge

acquisition (MKA) is then investigated with emphasis on the construction of KBFEs and the KASTLE project. Finally, some of the psychological and philosophical issues concerning the relationship between Mathematics and mathematicians (and mathematical thought processes) are discussed.

2 A Framework for the Systematisation of Mathematical Knowledge

In the same way as physical phenomena require models (specific instances of theories) to explain them, Mathematics requires a theory of mathematical knowledge. Furthermore, in the same way that models are derived from general theories, a theory of Mathematical Knowledge needs to be described in terms of a theory of knowledge itself. These requirements are shown in figure 1, which itself is based on the pragmatic theory of knowledge adhered to by Lewis, Pierce and Popper (see [43]).

The theory (or framework) for describing Mathematical Knowledge proposed here is similar to the theory of knowledge proposed by Addis for the general process of knowledge elicitation (see [2]). The theory is that mathematical systems can be described in terms of three different levels, each of which also interacts with mathematicians (see figure 2). The three levels are:

1. The underlying static mathematical knowledge.
2. The dynamic mathematical process (acting on the static mathematical knowledge).
3. The control of the mathematical process, which will here be called *heuristics* - see [39] (although the term heuristics is sometimes also used for either the discovery/informal description of the process or the general methodology employed).

This framework can be viewed as the initial stage of KA to be applied to Mathematics as a whole (it could be considered to be a mediating meta-knowledge representation for Mathematics - see [30]) and forms a basis for the study of Artificial Intelligence (AI) *within* Mathematics. The framework is applied in the following section.

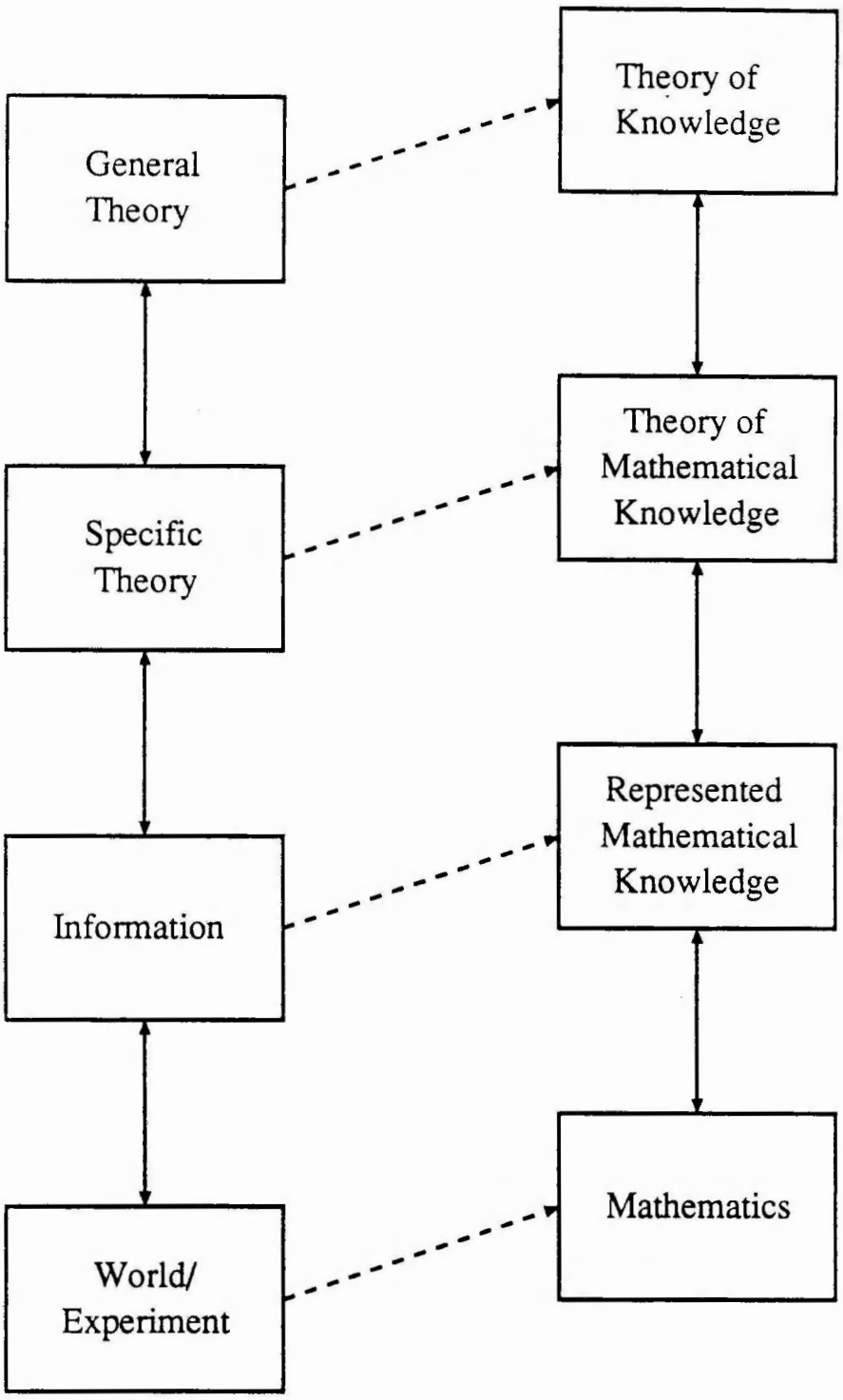


Figure 1: An Analogy between Science and Mathematical Knowledge

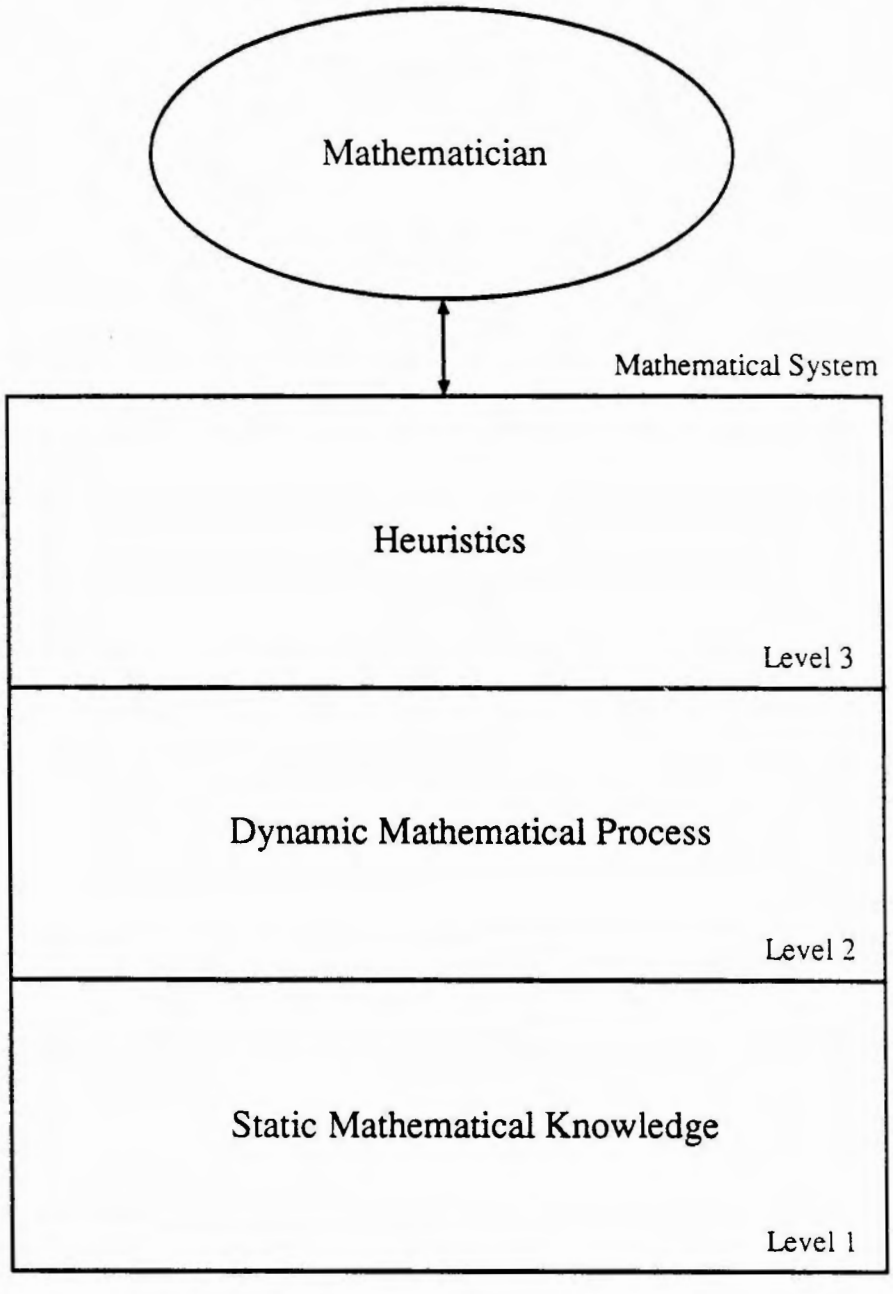


Figure 2: A Framework for the Systematisation of Mathematical Knowledge

3 Application of the Framework

3.1 Introduction

The purpose of this section is to describe Mathematical Knowledge in terms of the framework proposed above. The framework is first applied to Mathematics in general, then to the three sub-domains of Pure, Applied and Computational Mathematics, with discussions of the systems that are currently available. Finally, two more advanced areas of mathematical research are described and potential systematisations are discussed.

3.2 Mathematics in General

3.2.1 Background

Several 'natural language' descriptions of the nature and development of Mathematics already exist (e.g. see [37], [15] and [26]). Pseudo-formal representations of Mathematical Knowledge are given in [44] and [61]. The mathematical structure of a large number of physical theories has been investigated in depth by Tonti (see [75]).

3.2.2 Static Knowledge

Examples of organised static Mathematical Knowledge are:

- mathematical dictionaries - e.g. the recent new release of the Encyclopedic Dictionary of Mathematics, which classifies Mathematics into 450 subareas and represents several hundred years of effort (see [27]);
- mathematical bibliographies - there is currently a German project to provide a database of about 100,000 papers on Mathematical Logic; and
- mathematical tables - e.g. [1].

It will not be long before such static knowledge organisations will be available on-line widely and cheaply. The reign of manuscript organisations in this field is nearly over. On-line mathematical databases offer power and speed with the potential for conversion into computerised systems (see [51]).

3.2.3 Mathematical Process

The process of doing Mathematics in general has not been addressed in any great depth because of the historical emphasis on Mathematics as being a body of knowl-

edge, as already mentioned. This issue is discussed in the later section on Mathematics and mathematicians.

Several mathematical research programmes have been proposed and undertaken in the history of Mathematics, for example:

1. The Erlangen Programme - an exploration of the relationship between geometric structures and group theory proposed by Klein in Erlanger in the 1870's (see [35]). This programme is fairly unknown, but is significant to the systematisation of Mathematical Geometry (see [42]). The programme was also recently proposed by Franksen in relation to data structures (see [21]). A generalised Erlangen Programme could also form the basis for an analysis of Mathematical Knowledge by means of identifying the initial objects, the invariant objects, the types of equivalence and the transformations involved in different mathematical theories (see [62]) similar to (but less formally than) Category Theory.

2. The Hilbert Programme - a series of problems set by Hilbert around the turn of the century (see [26]). This programme had a significant impact on the formalisation of mathematical knowledge and the eventual creation of the computer.

3. The Borbaki Programme - a more recent detailed classification and description of Mathematics by a group of French mathematicians.

The cyclical nature of the mathematical research process has been by described by Vella - see [78] and figure 3.

3.2.4 Heuristics

The nature of heuristics has been addressed by Lenat in [39]. Mathematical Heuristics have been investigated informally by Pólya (see [56] and [57]), who has also discussed the use of plausible reasoning within Mathematics (see [58]). However, there does not exist a formal or pseudo-formal theories for Mathematical Heuristics at present. The problem is even worse than the lack of understanding of the mathematical process. An new paradigm is required which unifies the symbolist and connectionist paradigms of AI and applies them to Mathematics, the beginnings of which are seen in the COPYCAT system (see [45]).

However, certain areas of Mathematics do have more formally defined heuristics (e.g. large scale optmization problems) as will be described to an extent under the different headings below.

3.3 Pure Mathematics

The framework is applied to Pure Mathematics as follows:

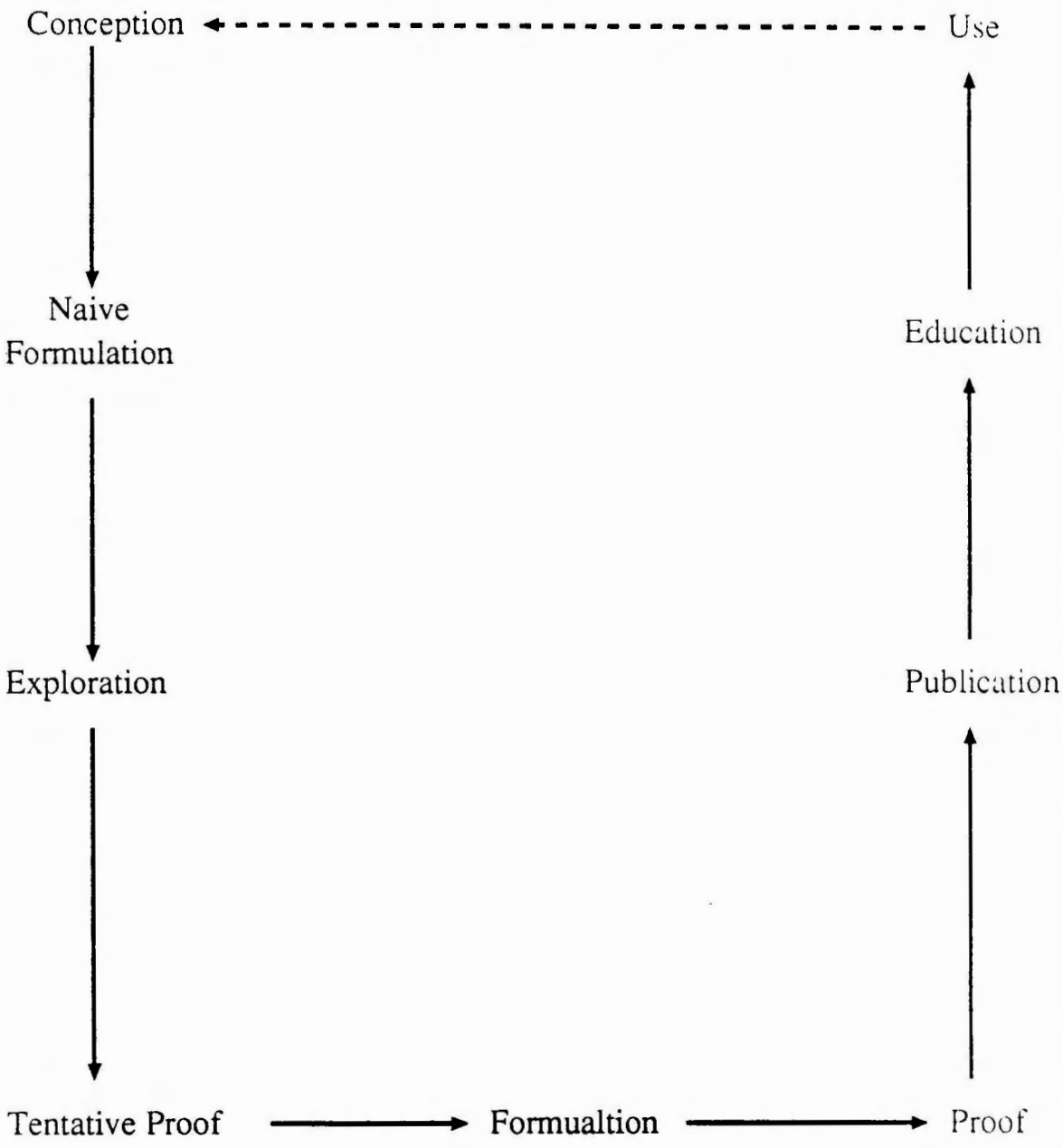


Figure 3: Vella's Diagram of the Mathematical Process

Static Knowledge: structures/axioms (e.g. the Peano Axioms for arithmetic over the natural numbers); useful derived concepts (e.g. prime numbers) and theories (e.g. that there is no largest prime).

Mathematcal Process: mathematical abduction (the creation of new structures and concepts - see [38]); and theorem-proving (see [12]).

Heuristics: Metamathematics (a semantic description of mathematical language - see [34] and [72]); and meta-level reasoning (see [52]).

Current pure mathematical systems available include the following:

Logical Languages e.g. Prolog - see [14];

Theorem-Provers e.g. MUSCADET (see [52]), which uses meta-level reasoning as well as object level reasoning (allowing for a degree of automated heuristics);

Algebraic Manipulation Packages e.g. Miranda - see [76]);

Pure Mathematics Discovery Systems There was considerable excitement in this field with the production of Lenat's systems: AM (see [38]) and EURISKO (see [40]). However, these systems did not appear to live up to their promise (see [60] and [41]) with many lessons being learnt about AI research in Mathematics (see [59]), but little further progress.

The current trend in this area is towards Pure Mathematics support environments, such as the Mathematics Understander which is currently being developed (see [22]).

3.4 Applied Mathematics

The framework is applied to Applied Mathematics as follows:

Static Knowledge: specific models and physical theories, e.g. the modelling of hydrodynamical systems in [5], which also contains interesting relevant discussions on the Mathematical Modelling process in this particular area.

Mathematcal Process: the process of Mathematical Modelling as illustrated by the examples in [3].

Heuristics: a theory of the process of Mathematical Modelling (e.g. [4]). This has not yet been recognised as being important and Mathematical Modelling is sometimes lowered to simply the solution of systems of Differential Equations (e.g. the main thrust of [73]).

The current Applied Mathematics systems available very limited (discounting standard computer simulations) and are as follows:

Computerised Mathematical Modelling Systems: e.g. the GOLEM system (see [11]). These are quite basic at present, only allowing for a degree of qualitative reasoning. (Also see [12].)

Mathematical Systems Theory: e.g. Tonti's diagrams and mathematical theory of physical theories (see [75]), Kron's method of tearing (see [20]), dimensional analysis and the well-posedness of mathematical problems. (Also see [79].)

3.5 Computational Mathematics

Computational Mathematics refers to mathematical procedures which have been computerised (e.g. matrix operations in Linear Algebra). The framework does not fit as well here as there is an ambiguity between algorithms and the mathematical process. This ambiguity is borne out below.

The framework is applied to Computational Mathematics as follows:

Static Knowledge: algebraic/statistical structures *or* algorithms for Numerical Analysis and Statistics (see [7]).

Mathematcal Process: the implementation of algorithms (feasible computation) *or* the classification and selection of algorithms.

Heuristics: control of algorithms *or* classification theory with its application to the structure of Computational Mathematics and the control of algorithm selection procedures (e.g. rule firing and conflict resolution for expert systems).

Current systems available are as follows:

Software Libraries: for example, the NAG Fortran Library (see [50]) is a large collection of efficient, tested algorithms for solving problems in many aspects of Numerical Analysis and Statistics, corresponding to about 2000 years of human effort.

KBFEs: these are knowledge-based systems that provide an improved computing environment for the use of numerical software. Descriptions of the KBFE research at NAG are given in [13] and [69].

Problem-Solving Environments: (PSEs) these are environments incorporating several components in addition to KBFEs (e.g. user problem representation, induction on user-supplied data, algebraic manipulation) - see [18]. Long-term research into PSEs is being carried out in the FOCUS and NEXUS projects (see [19] and [23] respectively).

3.6 Advanced Mathematical Areas

3.6.1 Introduction

There are many areas of advanced research within Mathematics. However, in terms of the systematisation of mathematical knowledge, certain areas are more difficult and challenging by virtue of their compound nature.

Two such advanced areas of current active research are Computational Fluid Dynamics (CFD) and Nonlinear Systems (NLS) Theory. These are described below:

3.6.2 Computational Fluid Dynamics

CFD (e.g. see [46]) is a highly sophisticated area of mathematical research involving Pure Mathematics (e.g. in terms of the Calculus and the analysis of algorithms); Applied Mathematics (in terms of forming the appropriate mathematical model of fluid flow); and Computational Mathematics (in terms of computing approximate solutions to analytically insoluble problems on a vast scale).

A summary of the concepts involved is given below:

1. The topology and geometry of the physical domain.
2. The representation of complex physical phenomena (such as shock-waves, vortex sheets, boundary layers, chemical reactions and turbulence) in a mathematical model.
3. The representation form of the equations of motion and the corresponding technique (e.g. integral form and finite volume techniques or weak form and finite elements).
4. The discretisation process (e.g. mesh regularity, data extrapolation).
5. Computational accuracy, convergence, efficiency, reliability and robustness.
6. Adaption and refinement of the solution (e.g. mesh adaption, mesh enrichment and shock capturing).
7. Visualisation of the solution (e.g. 3D colour graphics).
8. Validation of the model (e.g. comparison with experiment).

CFD is still a relatively unsettled area of mathematical research with many competing theories and methodologies. However, computers have already been used in automated mesh generation and sophisticated solution visualisation techniques. The trend is towards increasing systematisation within CFD, but this subject is seldom addressed directly; CFD expertise is still regarded as a 'black art'.

3.6.3 Nonlinear Systems Theory

NLS (see [74]) is here used to describe mathematical research in the following areas:

- the modelling of complex dynamical systems;
- chaos, catastrophe theory and the canonical form of instabilities; and
- fractal geometry, self-symmetry and non-integer dimensions.

NLS has already brought in a new research paradigm in Pure Mathematics by its use of experimentation and visualisation of complex set theory. Chaos also promises to be a rival theory to Quantum Mechanics and the very principle of causality. The systematisation of NLS is still at a very early stage. NLS appears to be a completely different kind of Mathematics.

4 KA and the KASTLE Project

4.1 Inroduction

This section describes an example of the MKA process: the production of a KBFE for the NAG Fortran Library for the KASTLE[1] project.

The process of KA within Mathematics if first discussed. After this, the KASTLE project is described, followed by a description of the different KA processes involved in the production of the KASTLE KBFE system and the different knowledge representations produced.

4.2 Mathematical Knowledge Acquisition

Mathematics appears, at first, to be a very appropriate domain for KA because of the vast amount of formal knowledge already in existence and the detailed understanding of mathematical reasoning expressed in Mathematical Logic. In fact, Mathematics has been defined by some to be the 'Science of Representation' (see [78]).

However, as already indicated, the mathematical process is not well understood. Also, the knowledge elicitation process in Mathematics is fairly unrepresentative, with greater emphasis needing to be placed on knowledge from documents, rather than experts; and the need for the knowledge engineer to become an expert himself and to acquire a complex mental model (see [31]) of the knowledge domain. Furthermore, MKA is hindered by the 'witchdoctor' status attributed to mathematicians and the reluctance of mathematicians themselves to learn anything from 'soft' sciences, such as Cognitive Science. However, times are changing.

[1]Knowledge Assisted Selection Tool for Library Environments

Knowledge Elicitation in Mathematics has been discussed by Johnson in [28], who suggests the use of personal constructs for knowledge representation within Mathematics in accordance with Kelly's theory (see [32]). Knowledge elicitation techniques tend to be unstructured, problem-solving interviews (e.g. proving theorems - see [52]) or introspective analysis (e.g. see [55]). There has been significant research undertaken into the cognitive processes involved in Mathematics for children (e.g. see [70]) but little for adults (also see the section on Mathematics and mathematicians).

4.3 The KASTLE Project

KASTLE is a DTI/SERC Teaching Company project between NAG and the Royal Military College of Science, Shrivenham. the project is nearing completion. Its objective is to produce a KBFE to the NAG Fortran Library (see [50]). The rapid prototyping methodology has been employed.

The most recent prototype to be delivered was P3 (see [71]). The final prototype, P4 (see [53]), is currently under development. P3 had the following features:

- SunView[2] WIMP[3]-based user interface.
- Expert routine selection and validation for part of the Fortran Library knowledge domain (Ordinary Differential Equations and Simultaneous Linear Equations) using attribute-and-values form-filling (see examples 1 and 2 in the appendix). Keyword routine selection was also provided for the whole Library.
- Display of manual example programs for calling Library subroutines.
- Trial hypertext-based help system for the form-filling knowledge domain.

P4 will contain the following features:

- Motif[4] X-Windows[5] WIMP-based user-interface.
- Expert routine selection, validation and local justification for a restricted class of Library routines using a simpler keyword-based approach.
- Example programming environment allowing the viewing, editing and execution of manual example programs.
- A more extensive and better designed hypertext help system.

[2]SunView is a registered trademark of Sun Microsystems Incorporated
[3]Windows Icon Menus Pointer
[4]Motif is a registered trademark of the Open Software Foundation, Incorporated
[5]The X Window System is a registered trademark of the Massachusetts Institute of Technology

Both P3 and P4 have been written in Prolog (see [14]).

The KA aspects of the design of these prototypes is described in the following subsections. A general requirement of the knowledge acquisition process has been to make it as separate as possible from the coding and user interface design of the KBFE being constructed.

4.4 Library Classifications

Library classification concerns the subdivision of individual Library routines into separate areas of Numerical Analysis and Statistics. The basic classification model is a tree, but this is often extended according to the context and requirements (the classification of mathematical knowledge is a more general issue which is beyond the scope of this paper).

The best example of the classification of numerical software is the Guide to Available Mathematical Software (GAMS - see [10]) which classifies all the existing large numerical software libraries. The classification used (see [9]) is basically tree-based and both the classification and the software guide are updated in roughly a five year cycle. The classification has also been turned into an on-line system (see [8]).

The process of classifying numerical libraries is quite complex. The need to maintain historical structures and representations for the sake of the existing user base makes reclassification very difficult. An important feature of library classifications therefore is extendability and general flexibility. Another important factor is the difference between on-line and manuscript representations. On-line classifications can have more overlap and are not restricted to a sequential representation (although the underlying information is still likely to be stored sequentially).

The classification used in the KASTLE project involved an initial reclassification of the Fortran Library in 'sections' (see [64]). However, following on from expert knowledge evaluation, an extended chapters classification (turning the Library chapters into independent sub-contexts by the repetition of Library routines from other chapters) was adopted for P4 (see [63]).

The knowledge elicitation technique employed was unstructured interviews and tasks for company-based experts along with assimilation of existing classification knowledge.

4.5 Automated Keyword KA

The keywords used for routine selection in P3 were generated by the use of database and concordance tools along with Fortran programs and editor macros. The knowledge source was short English descriptions of the Library routines. These descriptions were transformed into Prolog facts that related keywords (and key 'phrases') to different Library routines. The process was very complicated and required consider-

able refinement by hand (e.g. deriving the phrases). The method was dropped in P4 because the quality of the knowledge produced was unacceptable due to the routine descriptions not having been written with global consistency and completeness in mind.

4.6 Knowledge Representation Language

4.6.1 Introduction

The need for an mediating knowledge representation language (KRL) in knowledge-based system design is well known (e.g. see - [30]). The KRL used in the KASTLE project has many facets and has been modified significantly during the design of different prototypes.

4.6.2 P3 Keyword Section Selection

The top level selection process in P3 was based on keywords with a 'distinction' relation which allowed the filtering down of a list of sections and the screen presentation of a list of relevant new keywords on the interface screen. This process has been adopted in P4 for both the top level selection of chapters and the lower level selection of routines, as described below. The KA process for the section keywords was similar to the P3 form-filling KRL KA as described below, but was somewhat less structured at the testing and evaluation stage. However, expert evaluation of this knowledge did suggest that its quality and presentation was acceptable.

4.6.3 Form-Filling KRL

The routine selection process in the P3 system was based on the form-filling user interaction model. Users were required fill in in attribute values and pass down a tree hierarchy of forms. The expert system then performed pattern matching on the user's problem description and the routine facts bases.

The KRL for P3 (see [65]) therefore contained the following types of knowledge:

1. A high level form management language to control the input of the user's attribute values (see example 1 in the appendix) and to specify the hypertext descriptions of the attribute-value dialogue.

2. A knowledge base with attribute conversion and rule firing and matching commands (see example 2 in the appendix). The rule firing mechanism was quite simple and did not require any conflict resolution.

3. A facts base of different instances of routines with corresponding attribute value assignments (similar to example 5 in the appendix).

4. Validation knowledge for the selection of a given routine (returning a list of attribute and value requirements). This KRL was designed but not actually used in the production of the system's knowledge as its conversion into Prolog proved to be too complicated.

KA for the P3 form KRL involved:

- Initial KRL design and 'template' KA by knowledge engineers (it is much easier for experts to work from low quality knowledge than nothing at all).
- Problem-solving tasks and follow-up interviews with Library experts. These were characterised by an initial stage of establishing a rapport with the experts and teaching them the knowledge framework in which to work.
- Knowledge debugging, refinement and evaluation against a test suite of example problems with the assistance of Library experts and a knowledge testing environment (allowing the editing and running of single tests or a test batch with comparison of output in a WIMP-based environment).

4.6.4 P4 Keyword KRL

Following on from the evaluation of the automated keyword KA in P3, it was decided that different types of keyword needed to be identified for P4 (see [66]. The current classification is:

Structures - things that routines work on or produce (e.g. 'equation' or 'matrix') or concepts to do with computing (e.g. parallel architectures).

Activities - underlying mathematical operations, things that individual routines do or processes achieved by a sequence of routines or a class of routines.

Requirements - user requirements to distinguish between different routines that do not fit into the above categories (e.g. simplified parameter lists).

The two level selection process of chapters and routines used in P3 has been retained, but the transition between the levels has been smoothed (or made 'seamless'), the details of which are beyond the scope of this paper. The same keyword KRL was used for both chapter and routine selection. The KRL contains the following types of knowledge:

Chapter Facts Base: structures, activities and requirements for each chapter in the Library with a distinction between chapter-level and routine-level keywords used in the seamless transition process (see example 3 in the appendix).

Presentation Knowledge: commands to control which keywords to present on the screen initially and then after other keywords have been chosen along with keyword aliases (see example 4 in the appendix); again, the details of this process are beyond the scope of this paper.

Attribute-Value Knowledge: rules that disallow the user to specify certain contradictory problems that the system would otherwise accept (e.g. specifying the coefficients in a Linear Equations problem to be both real and complex, even though certain routines can handle both cases). These rules, as their name suggest, correspond to the attribute-value model used in P3 and are required due to the extra flexibility implicit within the keyword KRL structure.

Routine Facts Bases: typed keyword lists for routines within a each extended chapter of the Library (see example 5 in the appendix). A distinction is made between solver routines (used to solve problems) and auxiliary routines (used in conjunction with solver routines). The keyword lists for the solver routines are divided into two to allow for routine selection validation. Weighting rule knowledge may also be included.

The KA process for producing the P4 keyword KRL will follow roughly the same lines as for the P3 section-level keywords and routine-level form KRL. However, it is anticipated that the use of more structured procedures and the assistance of a more extensive knowledge development environment will take some of the 'black art' out of the process (especially in the refinement stage).

4.7 Hypertext KA

P3 was the first prototype in the KASTLE project to contain a hypertext process. P3 hypertext was based around the description of different concepts related to Numerical Analysis and Statistics. The skills required for hypertext authoring are different to the authoring of normal sequential text (see [80]).

Several lessons were learnt during and after the experience of designing the hypertext knowledge, leading to the following KA methodology being adopted in P4 (also see [67] and [68]):

Concept Typing: the activities-structures-requirements keyword types used in the P4 keyword selection KRL are also employed in the P4 hypertext with the addition of the type 'System' to be used for concepts to do with the KBFE itself (e.g. 'text buffer') for use in an on-line hypertext help system.

Concept Naming: the keywords themselves have consistent names with the keyword selection knowledge and their internal representation has been made as near as possible to their external names (as far as is possible for Prolog

atoms). Extra keywords are required for system concepts and also for intermediate and abstract concepts not useful for routine selection (such as functional equations). The naming of keywords is a very important aspect of the hypertext KA process - time must be spent on identifying the concepts before their description can be written.

Concept Structuring: the keywords are placed within an extended tree hierarchy (see example 6 in the appendix): the basic tree is given by the PARENT and CHILDREN designations; the extensions are given by the RELATED_... designations. This allows for a depth-first sequential listing of the hypertext.

Description Authoring: the production of concept description is, in fact, fairly straightforward after the concepts have been named and structured properly. The hypertext keys within the descriptions are made consistent with the relations in the hypertext structure. The descriptions appear to be derived very easily from the knowledge engineer's mental model (see [31]) of the meaning of the concepts and their inter-relationships. This mental model is facilitated by the production of an intermediate knowledge representation of concept trees by means of a graphical tree editor.

It is not appropriate for the different stages identified above to be fully separate. For example, a degree of refinement of the concept structure and concept names (an even possibly the concept types) is required during the concept description authoring.

4.8 Knowledge Conversion and Implementation

The different forms of KRL used need to be converted into a system-readable representation. This representation is in Prolog and is less accessible to knowledge engineers and experts. The conversion process has been written in Prolog using Definite Clause Grammars (see [54]) and is available to the knowledge engineer within the knowledge testing/development environment.

5 Mathematics and Mathematicians

5.1 Introduction

In this section, the relationship between Mathematics and mathematicians is briefly explored with reference to Philosophy, Psychology and mathematical creativity.

5.2 Philosophical Aspects

Historically, the explicit notion of mathematicians has been almost totally excluded from the Philosophy of Mathematics, with its emphasis being on the three main

philosophical stances of Platonism, Formalism and Intuitionism (see [26]). However, Kitcher (see [33]) argues about how the development of Mathematical knowledge has been affected by the experiences of our ancestors.

Computer science, cognitive science and knowledge engineering is having a growing influence on mathematicians, encouraging them to communicate more widely and to work with the assistance of computer support environments and AI systems (see [36] and [29]).

Tymoczko argues the importance of the mathematical community within Mathematics (see [77]). For example, the acceptance or rejection of various well-known theorems in Pure Mathematics (such as Fermat's last theorem) emphasises the rôle of the social process within the development of Mathematical Knowledge. Theorems need to be presented coherently and to a discerning audience as well as being proved. As presentation skills become increasingly important, the social aspect of Mathematical Knowledge development will increase.

5.3 Psychology in Mathematics

The relationship between Mathematics and Psychology has been investigated by the Birkhoff in [6]. The mathematical thought processes can be summarised as follows (also see the introduction to [15]):

Abduction/Formulation - the process by which problems are formulated, hypotheses are made and new concepts are devised.

Deduction - the process of following through an argument by the application of rules of inference.

Abstraction - the creation of more general abstract structures and theories from existing structures and theories.

Analysis and Synthesis - splitting problems up and putting ideas together.

Analogical Reasoning - applying a reasoning process in one domain to a similar problem in another domain.

Spatial Reasoning - for example, the use of iconic representations (such as Venn Diagrams in Set Theory) as an aid to deduction and mental models.

The use of cognitive processes in Mathematics has been investigated in [70]. However, this book relates almost completely to children. On the other hand, a characterisation of the mentality of mathematical experts has been given by Dehn in [16].

5.4 Mathematical Creativity

It is a moot point whether mathematical creativity exists as a separate subconscious thought process. Poincaré (see [55]) suggested that mathematical deductions came to him in the most obscure circumstances (such as: 'As soon as I stepped on the bus, the theorem came to me.'!). However, others argue that creativity and discovery come from concentrating on the right things (e.g. Pólya's imperative to: 'Look at the unknown!' in [56]). Muir recently distributed a survey on the subject (see [47]) based on the work by Hadamard (see [24]), with interesting (although not statistically significant) findings ([48]). This led to the IMA workshop, rather ambiguously named: 'How Mathematicians Work' (see [49]) held last year.

6 Conclusions

The vast amount of human effort that has been put into the development of Mathematics needs to be exploited by the power of the computer.

Mathematics needs to be regarded as a systematisable process that interacts with people (see figure 2), rather than as an impersonal body of knowledge. Mathematicians need to be treated (and also to regard themselves) as experts and not as witchdoctors, taking on an integrated rôle within the 'high tech' society.

These changes in perception will enivitably come and will facilitate the advancement of Mathematical Knowledge and the development of Mathematics within science.

References

[1] M. Abramowitz and I. A. Stegun, editors. *Handbook of Mathematical Functions.* Dover Publications, Incorporated, New York, 1965.

[2] T. R. Addis. A framework for knowledge elicitation. In *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems.* Reading University, September 1987.

[3] J. G. Andrews and R. R. McLone, editors. *Mathematical Modelling.* Butterworths, London, 1976.

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

[5] G. Birkhoff. *Hydrodynamics.* Princeton University Press, 1960.

[6] G. Birkhoff. Mathematics and psychology. *SIAM Review,* 11(4):429–469, October 1969.

[7] E. K. Blum. *Numerical Analysis and Computation: Theory and Practice.* Addison Wesley, Reading, MA, 1972.

[8] R. F. Boisvert. The Guide to Available Mathematical Software advisory system. *Mathematics and Computation in Simulation*, 31:453–464, 1989.

[9] R. F. Boisvert, S. E. Howe, and D. K. Kahaner. The Guide to Available Mathematical Software problem classification system. Technical Report NISTIR 90-4475, National Institute of Standards and Technology, Gaithersburg, MD 20899, November 1990.

[10] R. F. Boisvert, S. E. Howe, D. K. Kahaner, and J. L. Springman. Guide to Available Mathematical Software. Technical Report NISTIR 90-4237, National Institute of Standards and Technology, Gaithersburg, MD 20899, 1990.

[11] I. Bratko, S. Muggleton, and A. Varšek. Learning qualitative models in dynamic systems. In J. H. Johnson, S. McKee, and A. Vella, editors, *Artificial Intelligence in Mathematics.* Oxford University Press, to appear. Proceedings of IMA Conference, Strathclyde University, Glasgow, 3–5 April 1991.

[12] A. Bundy. *The Computer Modelling of Mathematical Reasoning.* Academic Press, 1983.

[13] J. Chelsom, D. J. Cornali, and I. Reid. Numerical and statistical knowledge-based front-ends research and development at NAG. In *The Second International Conference on Expert Systems for Numerical Computing*, pages 34–36. Purdue University, March 1990. Technical Report CSD-TR-963.

[14] A. Colmerauer. Prolog in 10 figures. *International Journal of Computation and Artificial Intelligence*, pages 487–499, 1983.

[15] R. Courant and H. Robbins. *What is Mathematics?* Oxford University Press, 1941.

[16] M. Dehn. The mentality of a mathematician. A characterization. *The Mathematical Intelligencer*, 5(2):18–26, 1983. Springer-Verlag, New York.

[17] P. Dowling and R. Noss. *Mathematics versus the National Curriculum.* Falmer Press, Basingstoke, 1990.

[18] B. Ford and F. Chatelin, editors. *Problem Solving Environments for Scientific Computing*, Amsterdam, 1987. Elsevier. IFIP Working Group 2.5 Working Conference, Sophia, Antipolis, 1985.

[19] B. Ford, S. J. Hague, R. M. J. Iles, and I. Reid. A framework for combining reasoning components with mathematical software. In J. H. Johnson, S. McKee, and A. Vella, editors, *Artificial Intelligence in Mathematics.* Oxford University Press, to appear. Proceedings of IMA Conference, Strathclyde University, Glasgow, 3–5 April 1991.

[20] O. I. Franksen. Kron's method of tearing. Presented at the Fourth Power Industry Computer Aided Application Conference, Clearwater, Florida, May 19–21, 1965.

[21] O. I. Franksen. Are data-structures geometrical objects? I: Invoking the erlanger program. *Systems Analysis, Modelling and Simulation*, 1(2):113–130, 1984.

[22] E. Furse. The mathematics understander. In J. H. Johnson, S. McKee, and A. Vella, editors, *Artificial Intelligence in Mathematics*. Oxford University Press, to appear. Proceedings of IMA Conference, Strathclyde University, Glasgow, 3–5 April 1991.

[23] P. W. Gaffney, C. A. Addison, B. Bjornestead et. al. NEXUS - towards a problem solving environment (PSE) for scientific computing. Technical Report CCS 86/9, Centre for Computer Science, Chr-Michelson Institute, Bergen, Norway, 1986.

[24] J. Hadamard. *The Psychology of Invention in the Mathematical Field*. Princeton University Press, New Jersey, 1945.

[25] A. Hart. *Knowledge Acquisition for Expert Systems*. Kogan Page, London, second edition, 1989.

[26] R. Hersh and P. J. Davis. *The Mathematical Experience*. Birkhäuser, Boston, 1981.

[27] Kiyosi Itô, editor. *Encyclopedic Dictionary of Mathematics*. MIT Press, London, second edition, 1987.

[28] J. H. Johnson. Knowledge elicitation in mathematics. In J. H. Johnson, S. McKee, and A. Vella, editors, *Artificial Intelligence in Mathematics*. Oxford University Press, to appear. Proceedings of IMA Conference, Strathclyde University, Glasgow, 3–5 April 1991.

[29] J. H. Johnson and M. Loomes, editors. *The Mathematical Revolution Inspired by The Computer*. Oxford University Press, 1991.

[30] N. E. Johnson. Mediating representations in knowledge elicitation. In D. Diaper, editor, *Knowledge Elicitation: Principles, Techniques and Applications*. Ellis Horwood Ltd., Chichester, 1989.

[31] P. N. Johnson-Laird. *Mental Models: Towards a Cognitive Science of Language, Inference and Consciousness*. Cambridge University Press, 1983.

[32] G. A. Kelly. *The Psychology of Personal Constructs*. W. W. Norton & Company, New York, 1955.

[33] P. Kitcher. *The Nature of Mathematical Knowledge.* Oxford University Press, 1984.

[34] S. Kleene. *Introduction to Metamathematics.* Van Nostrand, New York, 1952.

[35] F. Kline. Vergleichende Betrachtungen über neuere geometrische Forschungen. *Mathematische Annalen*, 43:63–100, 1893.

[36] D. E. Knuth. Computer science and its relation to mathematics. *Journal of the American Mathematical Society*, 81:323–343, 1974.

[37] E. E. Kramer. *The Nature and Growth of Modern Mathematics.* Princeton University Press, 1982.

[38] D. B. Lenat. *AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search.* PhD thesis, Stanford University, 1976. AIM-286, STAN-CS-76-570.

[39] D. B. Lenat. The nature of heuristics. *Artificial Intelligence*, 19:189–249, 1982.

[40] D. B. Lenat. EURISKO: A program that learns new heuristics and domain concepts, The nature of heuristics III: Program design and results. *Artificial Intelligence*, 21:61–98, 1983.

[41] D. B. Lenat and J. S. Brown. Why AM and EURISKO appear to work. *Artificial Intelligence*, 23:269–294, 1984.

[42] E. A. Lord and C. B. Wilson. *The Mathematical Description of Shape and Form.* Ellis Horwood, Chichester, 1984.

[43] A. G. J. MacFarlane. Interactive computing: A revolutionary medium for teaching and design. *IEE Computing & Control Engineering Journal*, pages 149–158, July 1990.

[44] S. MacLane. *Mathematics: Form and Function.* Springer Verlag, New York, 1986.

[45] M. Mitchell and D. R. Hofstadter. The emergence of understanding in a computer model of concepts and analogy-making. *Physica D*, 42:322–334, 1990.

[46] K. W. Morton and M. J. Baines, editors. *Numerical Methods for Fluid Dynamics III*, Oxford, 1989. Clarendon Press. Proceedings of Third International Conference on Numerical Methods in Fluid Dynamics, Oxford, March 1988.

[47] A. Muir. The psychology of mathematical creativity. *The Mathematical Intelligencer*, 10(1):33–35, 1988. Springer-Verlag, New York.

[48] A. Muir. Report on survey. Private Communication, 1988.

[49] A. Muir and D. Simpson. How mathematicians work. First IMA Workshop, Brighton Polytechnic, 1990.

[50] The Numerical Algorithms Group Limited, Oxford. *The NAG Fortran Library Manual*, mark 14 edition, 1990.

[51] K. Parsaye, M. Chignell, S. Khoshafian, and H. Wong. Intelligent databases. *AI Expert*, pages 38–47, March 1990.

[52] D. Pastre. MUSCADET: An automatic theorem proving system using knowledge and metaknowledge in mathematics. *Artificial Intelligence*, 38:257–318, 1989.

[53] M. Pengelly, J. D. Pryce, P. C. Samuels, J. Smith, and R. W. H. Whitmore. P4 specification. Internal Report of the NAG/RMCS Teaching Company project, Reference TCS/P4/34, January 1991.

[54] F. C. N. Periera and D. H. D. Warren. Definite clause grammars for language analysis - a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13:231–278, 1980.

[55] H. Poincaré. Mathematical discovery. In *Science and Method.* Nelson, 1914.

[56] G. Pólya. *How To Solve It: A New Aspect of Mathematical Method.* Princeton University Press, New Jersey, second edition, 1957.

[57] G. Pólya. *Mathematical Discovery: On Understanding, Learning and Teaching Problem Solving.* John Wiley & Sons, Inc., New York, 1962. volumes 1 and 2.

[58] G. Pólya. *Mathematics and Plausible Reasoning.* Princeton University Press, 1968.

[59] G. D. Ritchie. Learning from AM. In J. H. Johnson, S. McKee, and A. Vella, editors, *Artificial Intelligence in Mathematics.* Oxford University Press, to appear. Proceedings of IMA Conference, Strathclyde University, Glasgow, 3–5 April 1991.

[60] G. D. Ritchie and F. K. Hanna. AM: A case study in AI methodology. *Artificial Intelligence*, 23:249–268, 1984.

[61] P. C. Samuels. Developments in mathematical knowledge. Department of Computer Science, University of Reading, February 1989.

[62] P. C. Samuels. Representations for scientific and mathematical knowledge. Presented at international conference on 'Evolving Knowledge', Reading University, 1989, 1989.

[63] P. C. Samuels. Extended library chapters specification. Internal Report of the NAG/RMCS Teaching Company project, Reference TCS/P4/21, December 1990.

[64] P. C. Samuels. Fortran library sections subdivision with associated keywords: Vertsion 4. Internal Report of the NAG/RMCS Teaching Company project, Reference TCS/P3/70, June 1990.

[65] P. C. Samuels. The knowledge representation language for P3. Internal Report of the NAG/RMCS Teaching Company project, Reference TCS/P3/72, June 1990.

[66] P. C. Samuels. Chapter keywords structure. Internal Report of the NAG/RMCS Teaching Company project, Reference TCS/P4/38, February 1991.

[67] P. C. Samuels. P4 hypertext: Proposed knowledge engineer's representation. Internal Report of the NAG/RMCS Teaching Company project, Reference TCS/P4/39, February 1991.

[68] P. C. Samuels. Hypertext for computational mathematics. In J. H. Johnson, S. McKee, and A. Vella, editors, *Artificial Intelligence in Mathematics*. Oxford University Press, to appear. Proceedings of IMA Conference, Strathclyde University, Glasgow, 3–5 April 1991.

[69] P. C. Samuels. Knowledge-based front-ends for large numerical libraries. In *Proceedings of 13th IMACS World Congress on Computation and Applied Mathematics, 22–26 July 1991*, to appear.

[70] J. Sloboda and D. Rogers. *Cognitive Processes in Mathematics*. Oxford University Press, 1987.

[71] J. Smith and R. W. H. Whitmore. P3 user guide. Internal Report of the NAG/RMCS Teaching Company project, Reference TCS/P3/95, October 1990.

[72] A. Tarski. *Logic, Semantics, Metamathematics*. Oxford University Press, 1956.

[73] A. Tayler. *Mathematical Models in Applied Mathematics*. Oxford University Press, 1986.

[74] J. F. Toland and J. M. Ball. The mathematics of nonlinear systems. Proceedings of the SERC conference, University of Bath, 1–5 July 1991, to appear.

[75] E. Tonti. A mathematical model for physical theories. Technical report, Istituto di Meccanica Razionale del Politecnico di Milano, Piazza Leonardo da Vinci, 32 - 20133 Milano, Italy, 1972.

[76] D. A. Turner. An overview of Miranda. In *Research Topics in Functional Programming*. Addison-Wesley, 1990.

[77] T. Tymoczko. Making room for mathematicians in the philosophy of mathematics. *The Mathematical Intelligencer*, 8(3):44–50, 1986. Springer-Verlag, New York.

[78] A. Vella and C. Vella. Some reflections on artificial intelligence in mathematics. In J. H. Johnson, S. McKee, and A. Vella, editors, *Artificial Intelligence in Mathematics*. Oxford University Press, to appear. Proceedings of IMA Conference, Strathclyde University, Glasgow, 3–5 April 1991.

[79] B. H. Vorhees. Systems, models, methods, epistemic instrumentation. In J. P. van Gigch, editor, *Decision Making about Decision Making: Metamodels and Metasystems*. Abacus Press, Tunbridge Wells, Kent, 1987.

[80] N. Yankelovich, N. Meyrowitz, and A. van Dam. Reading and writing the electronic book. *IEEE Computer*, 18(10):15–30, October 1985.

Appendix: Examples of the KASTLE KRL

Example 1: P3 Form Management KRL

```
% ****** First Order Known Stiffness Form ******

BEGIN_FORM 2

  BEGIN_DETAILS

    NAME first_ord_kn_stiff SLOGAN 'First Order Known Stiffness'

    LEVEL 2

    ACTIVATED_WHEN stiff=[yes,no] AND order=one AND fn_vals=real

    CHILDREN [3 % comp_imp_or_stiff % ,4 % first_ord_non_stiff %]

  END_DETAILS

  BEGIN_ATTRIBUTES

    OFFER routine [simple,comprehensive]
      BEGIN_TEXT

      'Do you require a
       {simple_driver_routine,simple driver routine} or do you
```

```
        want to use a {comprehensive_routine,comprehensive routine}
        (with more parameters and sophisticated
        {error_estimation,error control} for
        {equation_stiffness,non-stiff problems} and a
        user-supplied {jacobian_matrix,Jacobian matrix} for
        {equation_stiffness,stiff problems})?'

      END_TEXT

    WHEN routine=comprehensive
    OFFER loc_err_est SLOGAN 'Local Error Estimate' [no,yes]
      BEGIN_TEXT

       'Do you require an
        {local_error_estimate,estimate of the local error}
        in the solution?'

      END_TEXT

  END_ATTRIBUTES

  BEGIN_MANAGEMENT

    SUGGEST routine=simple

    WHEN routine=comprehensive SUGGEST loc_err_est=no

  END_MANAGEMENT

END_FORM 2
```

Example 2: P3 Form Knowledge Base KRL

```
BEGIN_KNOWLEDGE_BASE 'Simultaneous Equations'

  % ********** General Routine Attributes **********

  FIRE method [b_b_or_g_p,least_sq_hom,nonlin_eqs]
  FORM_NAME
  EXACT_MATCH
```

```
FIRE coefficients [real,complex]
SIMPLE
MATCHING_TABLE
   [complex, real]           % routine
  [[YES,     20  ], complex, % \
   [40,      YES ], real,    %  } problem
   [20,      10  ], undef]   % /
% A complex routine can be used for a real problem by setting the
% imaginary part to be zero.
% A real routine can be used for a complex problem by equating real
% and imaginary parts in the problem and changing the variables
% accordingly.

% ********** Linear Equations Routine Attributes **********

WHEN method=[b_b_or_g_p,least_sq_hom]

  FIRE routine_memory [paged,either,unpaged]
  CONVERSION_RULE [mem_type] [problem_size]
    IF mem_type=undef THEN_ITS undef
    ELSE_IF mem_type=unpaged THEN_ITS unpaged
    ELSE_IF problem_size=[small,medium] THEN_ITS either
    ELSE_ITS paged
  MATCHING_TABLE
     [paged, unpaged]          % routine
    [[YES,   50     ], paged,  %  \
     [YES,   20     ], either, %   \
     [10,    YES    ], unpaged, %   } problem
     [5,     20     ], undef]  %   /

END_CONDITION method
```

Example 3: P4 Keyword Facts for a Single Library Chapter

```
CHAPTER c05
CHAPTER_ACTIVITIES
[solution_of_equations,roots_of_equations]
CHAPTER_STRUCTURES
[equation,nonlinear_equation,transcendental_equation,
 continuous_function]
ROUTINE_ACTIVITIES []
```

```
ROUTINE_STRUCTURES
[single_equation,simultaneous_equations,first_derivative]
ROUTINE_REQUIREMENTS
[forward_communication,reverse_communication,easy_to_use,
 comprehensive,given_interval,given_starting_value,
 check_derivatives]
```

Example 4: P4 Keyword Presentation Knowledge

```
% INITIAL display keywords

INITIAL
[data,equation,function,matrix,number,vector]

% INDEPENDENT keywords (not to be displayed until after initial
% selection)

INDEPENDENT
[cpu_time,curve,date,design,library_details,norm,regression_model,
 statistic,surface,time,variate]

% Keywords DEPENDENT on INITIAL display keywords

DEPENDENT data
[character_data,data_matrix,data_plot,data_table,multivariate_data,
 raw_data,series,univariate_data]

% Keywords DEPENDENT on INDEPENDENT keywords

DEPENDENT design
[block_factor,treatment_factor]

% Keywords DEPENDENT on other DEPENDENT keywords

DEPENDENT algebraic_equation
[linear_algebraic_equation,nonlinear_algebraic_equation]

% CROSS_DEPENDENT keywords

CROSS_DEPENDENT approximation
[curve,surface]
```

```
% Keyword ALIASes

ALIAS general_linear_model regression_model
```

Example 5: P4 Keyword Routine Facts Base

```
% An extended chapter solver routine

SOLVER e02gaf
PRE_CALLS []
POST_CALLS [f04ycf]
ACTIVITIES [least_squares_problems,overdetermined_problems]
           []
STRUCTURES [rectangular_matrix,real_matrix,one_right_hand_side,
            full_matrix,full_rank_matrix]
           [matrix_condition_number,many_right_hand_sides(40)]
REQUIREMENTS [approximate_solution,unpaged_memory,one_system]
             [paged_memory(20),many_systems(40)]

% Some auxiliary routines

AUXILIARY f01brf
ACTIVITIES [matrix_factorization]
STRUCTURES [square_matrix,real_matrix,sparse_matrix]
REQUIREMENTS [one_system]

AUXILIARY f01bsf
ACTIVITIES [matrix_factorization]
STRUCTURES [square_matrix,real_matrix,sparse_matrix]
REQUIREMENTS [many_systems]

AUXILIARY f01lhf
ACTIVITIES [matrix_factorization]
STRUCTURES [square_matrix,real_matrix,structured_matrix,
            block_diagonal_matrix]
REQUIREMENTS []
```

Example 6: P4 Hypertext Structure

```
ACTIVITY linear_algebra
PARENT activities
CHILDREN [linear_algebra_operations,eigenproblems]
RELATED_ACTIVITIES [solution_of_equations,arithmetic]

  ACTIVITY linear_algebra_operations
  PARENT linear_algebra
  CHILDREN [scalar_operations,vector_operations,matrix_operations,
            rotation,reflection,permutation,norm_evaluation]
  RELATED_STRUCTURES [linear_algebra_structure]

    ACTIVITY scalar_operations
    PARENT linear_algebra_operations
    RELATED_STRUCTURES [scalar]
```