Knowledge Acquisition (1992) **4**, 5–53

KADS: a modelling approach to knowledge engineering

B. J. WIELINGA, A. TH. SCHREIBER AND J. A. BREUKER

Department of Social Science Informatics, University of Amsterdam, Roetersstraat 15, NL-1018 WB Amsterdam, The Netherlands

This paper discusses the KADs approach to knowledge engineering. 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. Five basic principles underlying the KADS approach are discussed, namely (i) the introduction of partial models as a means to cope with the complexity of the knowledge engineering process, (ii) the KADS four-layer framework for modelling the required expertise, (iii) the re-usability of generic model components as templates supporting top-down knowledge acquisition, (iv) the process of differentiating simple models into more complex ones and (v) the importance of structure—preserving transformation of models of expertise into design and implementation. The actual activities that a knowledge engineer has to undertake are briefly discussed. We compare the KADS approach to related approaches and discuss experiences and future developments. The approach is illustrated throughout the paper with examples in the domain of troubleshooting audio equipment.

1. Introduction

This paper discusses results of a European research project commonly known as the KADS project (ESPRIT-I P1098). This project aimed at the development of a comprehensive, commercially viable methodology for knowledge-based system (KBS) construction. When the KADS project was conceived, in 1983, little interest in methodological issues existed in the AI community. The prevailing paradigm for building knowledge-based systems was rapid prototyping using special-purpose hard- and software, such as LISP machines, expert system shells etc. Since then, many organizations have become aware of the fact that KBS development from an organizational point of view does not differ much from the development of other types of information systems. Aspects of KBS development such as information analysis, application selection, project management, user requirement capture, modular design, re-usability etc, are similar to those encountered in conventional system development. Problems that frequently occur in conventional information system development projects are amplified in the case of KBS development. The wider capabilities of KBS technology allow more complex applications, which have a stronger impact on organizational structure than most conventional systems and often require a more sophisticated user–system interaction than is the case with conventional systems. Additionally, KBS development poses a number of problems of its own.

1042-8143/92/010005 + 49$03.00/0

An often-cited problem in KBS construction is the *knowledge acquisition bottleneck*. It turns out to be very difficult to extract the knowledge that an expert has about how to perform a certain task efficiently in such a way that the knowledge can be formalized in a computer system. The actual realization of a KBS often poses problems as well. The reasoning methods that are used in KBSs are not always fully understood. Although the AI literature abounds in methods and techniques for modelling reasoning processes, their description is not uniform and unambiguous. So, the need for a sound methodology for KBS development has become recognized over the last few years.

In this paper we will discuss the principles that comprise the framework on which the KADS methodology is founded and describe its main ingredients.

2. Views on knowledge acquisition

During the knowledge acquisition process the knowledge that a 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 *formalizing* the conceptualizations in such way that the program can use the knowledge. In this paper we will mainly focus on the interpretation and formalization 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 now reasonably well-understood (Breuker & Wielinga, 1987; Neale, 1988; Diaper, 1989; Meyer & Booker, 1991).

Traditionally the knowledge acquisition process was 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 these rules into the appropriate format. Several authors (Hayward, Wielinga & Breuker, 1987; Morik, 1989) have pointed out that this 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 at the problem or the domain in a way different from the knowledge engineer, asking for rules or similar knowledge structures and translating them into the knowledge representation language of the system does not work.

A different view of 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 the models is a means of coping with the complexity of the development process.

Constructing a KBS is seen as building a computational model of desired behaviour. This behaviour can coincide with behaviour 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 of a patient may be a good starting point for a description of the intended problem-solving behaviour of the KBS. However, a KBS is hardly ever the functional and behavioural equivalent of an expert. There are a number of reasons for this. Firstly, the introduction of information technology often involves new distributions of functions and roles of agents. The KBS may perform functions which are not part of the experts repertory. Secondly, the underlying reasoning process of the expert cannot often be explained fully. Knowledge, principles and methods may be documented in a domain, but these are aimed at a human interpreter and are not descriptions of how to solve problems in a mechanical way. Thirdly, there is an inherent difference between the capabilities of machines and humans. For example, in an experiment in a domain of configuring moulds (Barthélemy, Frot & Simonin, 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 is essentially 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 modelling 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.

In this paper we will adopt the modelling perspective of knowledge acquisition. We discuss the principles that underlie the KADS approach to building knowledge-based systems, namely:

(i) The introduction of multiple models as a means to cope with the complexity of the knowledge engineering process (Section 3).
(ii) The KADS four-layer framework for modelling the required expertise (Section 4).
(iii) The re-usability of generic model components as templates supporting top-down knowledge acquisition (Section 5).
(iv) The process of differentiating simple models into more complex ones (Section 6).
(v) The importance of structure-preserving transformation of models of expertise into design and implementation (Section 7).

Although a description of the use of KADS in practical KBS projects is outside the scope of this article, we look briefly at the actual knowledge engineering process (Section 8). We also compare the KADS approach to other approaches (Section 9). Finally we discuss experiences and future developments (Sections 10 and 11).

The approach is illustrated throughout the paper with examples, most of them in the domain of diagnosing and correcting malfunctions of an audio system.

3. Principle 1: multiple models

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, conceptualization and formalization. Knowledge engineers are thus faced with a jungle of possibilities and find it difficult to navigate through this space.

The idea behind the first principle of KADs is that the knowledge-engineering space of choices and tools can to some extent be controlled by the introduction of a number of *models*. A model reflects, through abstraction of detail, selected characteristics of the empirical system in the real world that it stands for (DeMarco, 1982). Each model emphasizes certain aspects of the system to be built and abstracts from others. 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 a number of models, namely (i) the organizational model, (ii) the application model, (iii) the task model, (iv) the model of cooperation, (v) the model of expertise, (vi) the conceptual model and (vii) the design model.

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.

3.1. ORGANIZATIONAL MODEL, APPLICATION MODEL AND TASK MODEL

In KADS we distinguish three separate steps in defining the goals of KBS construction, namely, (i) defining the *problem* that the KBS should solve in the organization, (ii) defining the *function* of the system with respect to future users (which can be either humans or possibly other systems) and (iii) defining the actual *tasks* that the KBS will have to perform.

In this section we discuss three models that address parts of this three-step process. The first two are discussed briefly as these are outside the scope of this article.

Organizational model

An organizational model provides an analysis of the socio-organizational environment in which the KBS will have to function. It includes a description of the functions, tasks and bottlenecks *in the organization*. In addition, it describes (predicts) how the introduction of a KBS will influence the organization and the people working in it. This last activity can be viewed as a type of *technology assessment* (de Hoog, Sommer & Vogler, 1990)). We have found (de Hoog, 1989; van der Molen & Kruizinga, 1990) that it is dangerous to ignore the impact of the interaction between the construction of a KBS and the resulting changes in the organization. Neglecting this aspect may lead to a system that is not accepted by its prospective users. It is also important to realize that the process of KBS construction itself can, by its nature (for instance, through extensive interviewing), change the organization in such a way that it becomes a "moving target" (van der Molen &

Kruizinga, 1990). The result may be that the final system is aimed at solving a problem that does not exist any more in the organization. We are convinced that the organizational viewpoint is important throughout the KBS construction process.

Application model
An application model defines what problem the system should solve in the organization and what the function of the system will be in this organization. For example, the daily operation and fault handling of an audio system can pose serious problems for people who are not familiar with or just not interested in more than the superficial ins-and-outs of such a system. A potential solution to this problem could be the development of a knowledge-based system. The function of this system would be to ensure that the owner of the audio system is supported in the process of correcting operational malfunctions of the audio system.

In addition to the *function* of the KBS and the *problem* that it is supposed to solve, the application model specifies the *external constraints* that are relevant to the development of the application. Examples of such constraints are the required speed and/or efficiency of the KBS and the use of particular hardware or software.

Task model
A task model specifies how the function of the system (as specified in the application model) is achieved through a number of tasks that the system will perform. Establishing this relation between function and task is not always as straightforward as it may seem. For example, consider a problem such as the medical care of patients with acute infections of the bloodstream. One approach to solve this problem is to perform the following tasks: (i) determine the identity of the organism that causes the infection and (ii) select, on the basis of that diagnosis, the optimal combination of drugs to administer to the patient. In real life hospital practice however, the recovery of the patient is the primary concern. So, if identification of the organism proves difficult, e.g. because no laboratory data are available, a therapy will be selected on other grounds. In fact, some doctors show little interest in the precise identity of the organism causing an infection as long as the therapy works. Stated in more general terms: given a goal that a system should achieve, there may be several alternative ways in which that goal can be achieved. Which alternative is appropriate in a given application depends on the characteristics of that application, on availability of knowledge and data and on requirements imposed by the user or by external factors.

With respect to the content of the task model, we distinguish three facets: (i) task decomposition, (ii) task distribution and (iii) task environment.

Task decomposition. A task is identified that would achieve the required functionality. This task is decomposed into sub-tasks. A technique such as *rational task analysis* is often used to achieve such a decomposition. We call the composite top-task a "real-life task", as it often represents the actual task that an expert solves in the application domain. The sub-tasks are the starting point for further exploration, such as the modelling of expertise and cooperation. A simple decomposition of a real-life task in the audio domain is shown in Figure 1.

Each separate task is described through an input/output specification, where the output represents the goal that is achieved with the task and the input is the

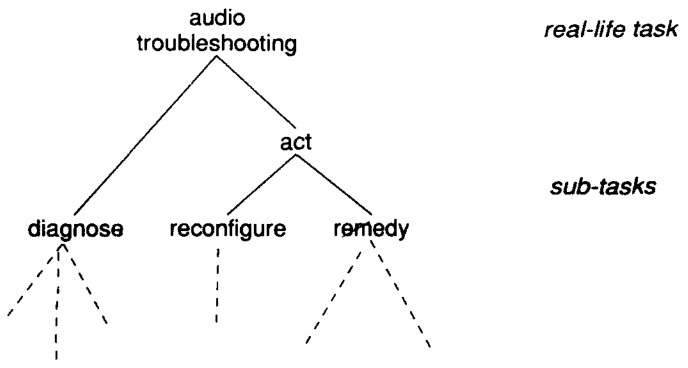


FIGURE 1. Task decomposition for the audio example.

information that is used in achieving this goal. What constitutes the goal of a task is not always self-evident. Even for a seemingly well-understood task such as diagnosis, it is not always clear what a diagnosis of a faulty system means. A diagnosis could be the identification of a subsystem (a component of an audio system) that malfunctions, or it could be a full causal model of how a malfunction came about. Similarly the result of a design task could be a detailed description of the structure of a system (e.g. a device for monitoring patients in an intensive care unit) or it could be a description of the functionality, structure and use of the device.
Task distribution. The task distribution is the *assignment* of tasks to *agents.* Example agents are the KBS, the user or some other system. The last two agents are called *external* agents. Given the task decomposition the knowledge engineer has to decide what sub-tasks to assign to the system and what tasks to the user. These decisions constitute essentially cognitive engineering problems (Roth & Woods, 1989): they should be made on the basis of an analysis of the user requirements and expectations, the knowledge and skills that the user has and the potential capabilities and limitations of the system.
Task environment. The nature of the task-domain itself usually enforces a number of constraints on how the task can be performed. We call the constraints the *task environment.* For example, the task environment of a support system for handling malfunctions in an audio system could consist of the following constraints:

(i) The KBS is not a physical part of the audio system.
(ii) It has no sensors to make observations (and thus depends on the user to do this).
(iii) It has no robot arm to perform reconfigurations and/on repairs (and thus again depends on the user to do this).
(iv) The KBS users will be novices, who are not expected to be able to understand technical terms or to examine the interiors of the audio system.

The constraints posed by the task environment influence both the scope and the nature of the models of expertise and cooperation (see below).

The task model and its role in specifying system–user interaction is discussed in more detail in de Greef and Breuker (1992, this issue).

3.2. MODEL OF COOPERATION

The task model consists of a decomposition of the real-life task into a number of primitive tasks and a distribution of tasks over agents. The model of cooperation contains a specification of the functionality of those sub-tasks in the task model that require a cooperative effort. These tasks can for instance be data acquisition tasks activated during problem solving or various types of explanation tasks. Such tasks are called *transfer* tasks, as they involve transferring a piece of information from the system to an external agent or vice versa.

There is thus a clear dependency between the model of cooperation and the model of expertise. Some of the sub-tasks will be achieved by the system, others may be realized by the user. For example, in a diagnostic task in the audio example, the system may suggest certain tests to be performed by the user, while the user will actually perform the tests and will report the observed results back to the system. Alternatively, the user may want to volunteer a solution to the diagnostic problem while the system will criticize that solution by comparing it with its own solutions.

The result is a model of cooperative problem solving in which the user and the system together achieve a goal in a way that satisfies the various constraints posed by the task environment, the user and the state-of-the-art of KBS technology. The modelling of cooperation is outside the scope of this paper, but is discussed in more detail in de Greef and Breuker (1992, this issue), de Greef, Breuker and de Jong (1988*a*) and de Greef and Breuker (1989).

3.3. 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—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 realized in the implementation. These two perspectives correspond to the distinction 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 categorization 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: *performing a test* and *carrying out a* reconfiguration. In the model of expertise, such interaction or *transfer* tasks are specified more or less as a black box (see Section 4.3). The detailed study of the nature of these transfer tasks is the subject of the modelling of cooperation.

As the model of expertise plays a central role in KBS development, its details are discussed extensively in Section 4.

3.4. CONCEPTUAL MODEL = MODEL OF EXPERTISE + MODEL OF COOPERATION

Together, the model of expertise and the model of cooperation provide a specification of the behaviour of the artefact to be built. The model that results from merging these two models is similar to that is called a *conceptual model* in database development. Conceptual models are abstract descriptions of the objects and operations that a system should know about, formulated in such a way that they capture the intuitions that humans have of this behaviour. The language in which conceptual models are expressed is not the formal language of computational constructs and techniques, but is the language that relates real world phenomena to the cognitive framework of the observer. In this sense conceptual models are subjective, they are relative to the cognitive vocabulary and framework of the human observer. Within KADs we have adopted the term "conceptual model" to denote a combined, implementation-independent, model of both expertise and cooperation.

3.5. DESIGN MODEL

The description of the computational and representational techniques that the artefact should use to realize the specified behaviour is not part of the conceptual model. 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 conceptual model specifications on the one hand and design decisions on the other, in our experience building a conceptual model without having to worry about system requirements makes life easier for the knowledge engineer.

The separation between conceptual modelling on the one hand and a separate design step on the other has been identified as both the strength and the weakness of the KADS approach (Karbach, Linster & Voß, 1990).

The main advantage lies in the fact that the knowledge engineer is not biased during conceptual modelling by the restrictions of a computational framework. KADs provides a more-or-less universal framework for modelling expertise (see the next section) and although computational constraints play a role in the construction of such models (cf. Section 6) experience† 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 conceptual model, is still faced with the problem of how to implement this specification. In Section 7 we discuss some principles that can guide the knowledge engineer in this design process.

Figure 2 summarizes the different roles which the conceptual model and the design model play in the knowledge engineering process. An observer (knowledge

† See Section 10 for an overview of applications developed with the KADS approach.

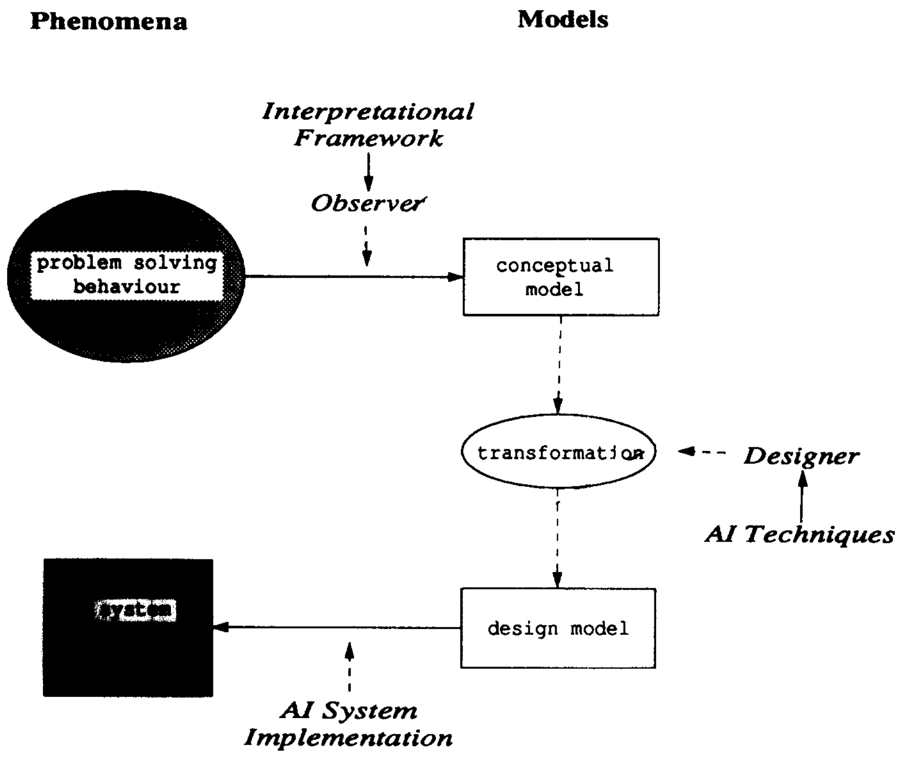


FIGURE 2. Role of the conceptual model and the design model in the knowledge acquisition process.

engineer) constructs a conceptual, knowledge-level, model of the artefact by abstracting from the 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 conceptual model is real-world oriented in the sense that, it is phrased in real-world terminology and can thus be used as a communication vehicle between knowledge engineer and expert. The conceptual model does not take detailed constraints, with regard to the artefact, into account. The desing model, on the other hand, is a model that is phrased in the terminology of the artefact: it describes how the conceptual model is realized with particular computational and representational techniques.

Figure 3 shows the dependencies between the models discussed in this section. Connections indicate that information from one model is used in the construction of another model. The actual activities in the construction process do not necessarily have to follow the direction from organization model to system. In fact, several life-cycle models have been developed, each defining various phases and activities in building these models. The first life-cycle model developed in KADS (Barthélemy, Edin, Toutain & Becker, 1987) was of the water-fall type. At the end of the KADS project, a new life-cycle was defined (Taylor *et al.*, 1989) based on the concept of a spiral model (Boehm, 1988).

The nature of knowledge engineering thus becomes a process that bridges the gap between required behaviour and a system that exhibits that behaviour through the

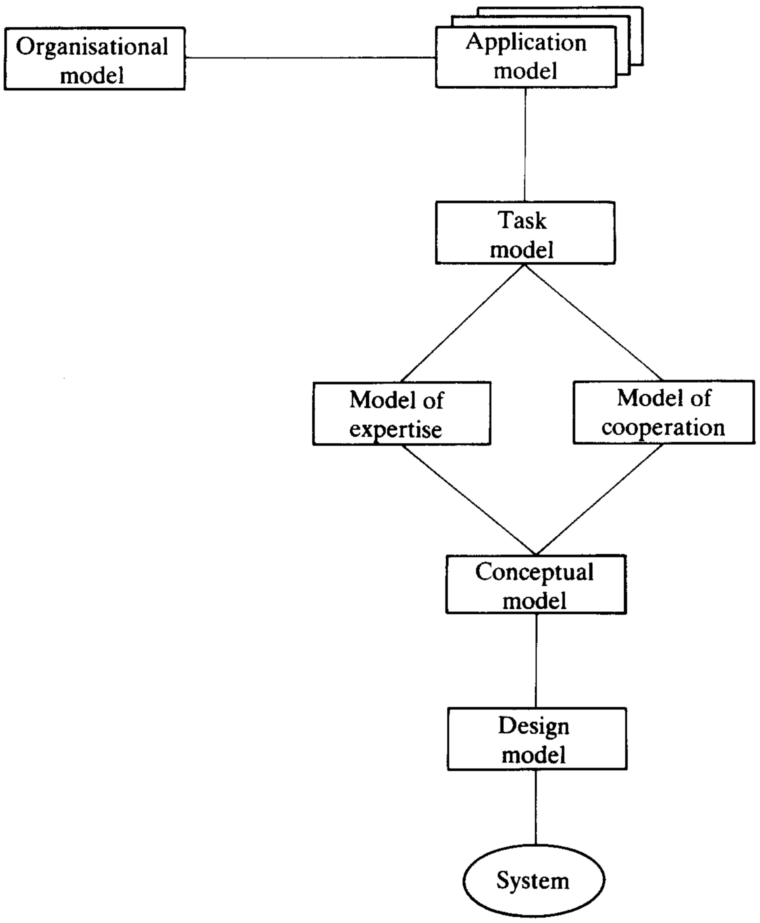


FIGURE 3. Principle 1: Partial models provide a decomposition of the knowledge-engineering task.

creation of a set of models. Summarizing, we can say that the KADS modelling view of knowledge engineering gives rise to a methodology that involves the construction of a variety of models in the course of the knowledge engineering process. Each model represents a particular view on the KBS. They allow the knowledge engineer to cope with the complexity of the knowledge engineering process through a "divide-and-conquer" strategy.

The remainder of this article focuses mainly on the model of expertise, as it plays such a central role in KBS development.

4. Principle 2: 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 and Breuker (1986), Hayward *et al.* (1987), Schreiber *et al.* (1988) and in Breuker and 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 organized into 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 knowledge can be analysed and described are based on epistemological distinctions: they contain different types of knowledge. We distinguish between, (i) static knowledge describing a declarative *theory* of the application domain (domain knowledge), (ii) knowledge of different *types of inferences* that can be made in this theory (first type of control knowledge), (iii) knowledge representing *elementary tasks* (second type of control knowledge) and (iv) *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 have proposed ways to increase the flexibility of control in expert systems. The work of Davis (1980) introduced explicit control knowledge as a means of controlling inference processes in a flexible way. In the NEOMYCIN system (Clancey, 1985*a*) different functions of knowledge are explicated by separating domain knowledge and control knowledge and by introducing an explicit description of the strategies that the system uses. 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.

4.1. DOMAIN KNOWLEDGE

The domain knowledge embodies the *conceptualization* of a domain for a particular application in the form of a *domain theory*. The primitives that we use to describe a domain theory are based on the epistemological primitives proposed by Brachman and Schmolze (1985): concepts, properties, two types of relations and structures.

Concept

Concepts are the central objects in the domain knowledge. 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.

Relation between property expressions
A second type of relation is the relation between expressions about property values. An expression is a statement about the value(s) of a property of a concept, e.g. `amplifier:power=on`.† Examples of this type of relation are causal relations and time relations. An example of a tuple of a causal relation in the audio domain could be:

```
amplifier:power-button=pressed CAUSES amplifier:power=on
```

Structure
A structure is used to represent a complex object: an object consisting of a number of objects/concepts and relations. For example, the audio system as a whole can be viewed as a structure, consisting of several components and relations (part-of, wire connections) between these components.‡

The choice of this set of primitives is, in a sense, arbitrary and probably somewhat biased by the types of problems that have been tackled with KADS. The problem is to find a subset that provides the knowledge-engineer with sufficient expressive power. One could consider including additional special-purpose primitives such as mathematical formulae. There is clearly a link here with research in the field of semantic database modelling (see, for an overview, Hull & King, 1987).

The 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 to use the term "schema" rather than "ontology" to stress the fact that the domain theory is the product of knowledge engineering and thus, does not necessarily describe 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 *conceptualization* (Genesereth & Nilsson, 1987; Nilsson,

† We use the shorthand ⟨`concept`⟩ : ⟨`property`⟩ for "the ⟨property⟩ of ⟨concept⟩".

‡ The term "structure" as used here should not be confused with the "structural descriptions" in KL-ONE.

§ The relation between property expressions corresponds to an axiom schema; structures correspond to a sub-theory.

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

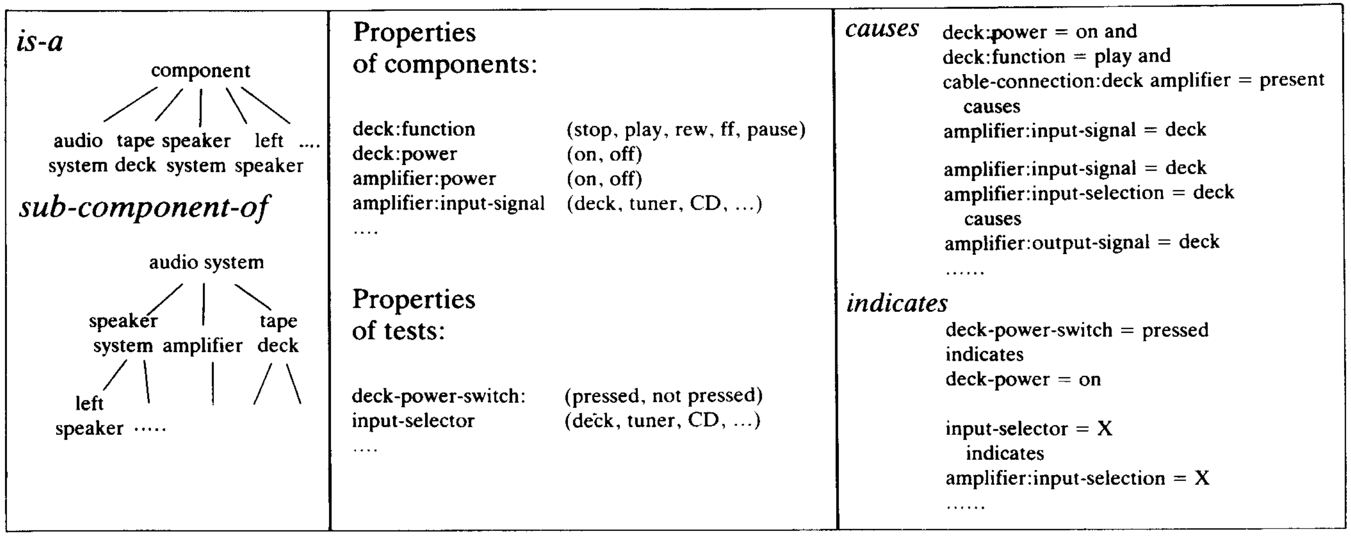
Primitive	Name	Description
Concept	component	The elements of the audio system
Relation between concepts	component IS-A component	Sub-type hierarchy of components of the audio system
Relation between concepts	component SUB-COMPONENT-OF component	Part-of hierarchy of components of the audio system
Property	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.

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 re-appear in similar domains and could be re-used (see Section 5 for a more detailed discussion of re-usability). The domain schema also provides convenient 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 of a simple domain theory for diagnosing faults in an audio system is shown in Table 1.† 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 *sub-component-of*. In addition, two relations between property expressions are defined: (i) a *causal* relation between state values of components, and (ii) an *indicates* relations between test values and state values.

Figure 4 shows some domain knowledge in the audio domain. The domain

† The description of the domain schema given here is rather informal. For example, nothing is said 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).



Properties of components:

deck:function	(stop, play, rew, ff, pause)
deck:power	(on, off)
amplifier:power	(on, off)
amplifier:input-signal	(deck, tuner, CD, ...)
....	

Properties of tests:

deck-power-switch:	(pressed, not pressed)
input-selector	(deck, tuner, CD, ...)
....	

causes

deck:power = on and
deck:function = play and
cable-connection:deck amplifier = present
causes
amplifier:input-signal = deck

amplifier:input-signal = deck
amplifier:input-selection = deck
causes
amplifier:output-signal = deck
......

indicates

deck-power-switch = pressed
indicates
deck-power = on

input-selector = X
indicates
amplifier:input-selection = X
......

FIGURE 4. Domain knowledge of the audio system using the schema described in Table 1.

knowledge description follows 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 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) showing 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 re-usability of domain knowledge.

4.2. 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 is fully 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 of the internal behaviour of the inference. One could look upon the inference as applying a simple theorem prover.

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 realized 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 entity that carries out an action in a primitive inference step is called a *knowledge source.*† A knowledge source performs 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 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 than one meta-class. For example, a

† The term "knowledge source" was inspired by Clancey's (1983) use of this term as a process that generates an elementary piece of information. Its intended meaning corresponds only roughly to the meaning of the term in blackboard architectures.

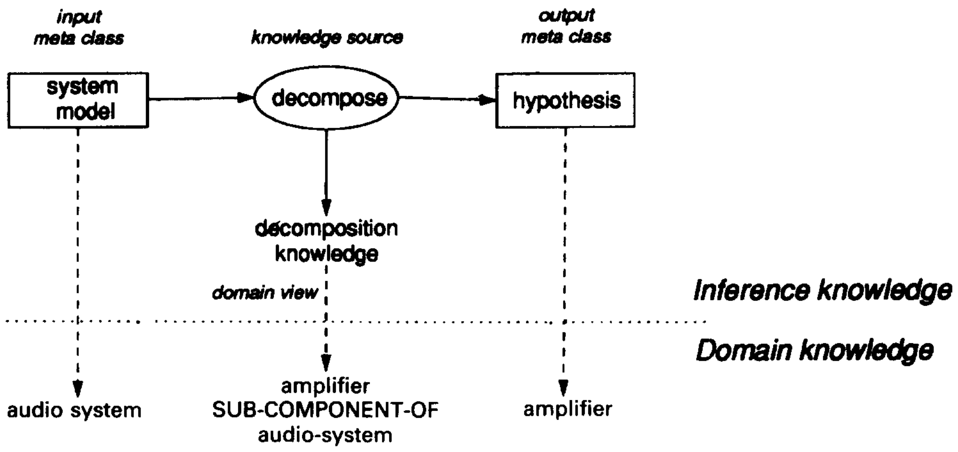


FIGURE 5. A primitive inference performing an decomposition action.

particular component of an audio system could play the role of a *hypothesis* at one point in time and the role of *solution* at some other instant. The name "meta-class" is inspired by the fact that it provides a "meta" description of objects in a domain "class".† An input data element of a knowledge source is referred to as an *input* meta-class; the output as an *output* metaclass. Each meta-class can be the 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 a "body of knowledge" by the knowledge source.

Figure 5 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 5 shows one applicable tupel of this relation.

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

knowledge-source *decompose*
 input-meta-class:
  `system-model→component`
 output-meta-class:
  `hypothesis→component`

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

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 knowledge in the domain theory, namely the SUB-COMPONENT-OF relation. In principle, however, there could be several of these mappings.†

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

(i) 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.
(ii) Domain knowledge that is used in more than one inference is specified only once. In this way, knowledge redundancy is prevented.
(iii) It provides a dual way to *name*‡ domain knowledge: both use-independent and use-specific. Knowledge engineers tend to give domain knowledge elements names that already reflect their intended use in inferencing and keep changing the names when their usage changes. We would argue that both types of names can be useful and should be known to the system—for example, for explanation purposes.
(iv) 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 conceptualizing a domain and specifying the problem solving process. We are convinced, however, that it is useful to *document* them at least 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

† 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.* (1989*b*).

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

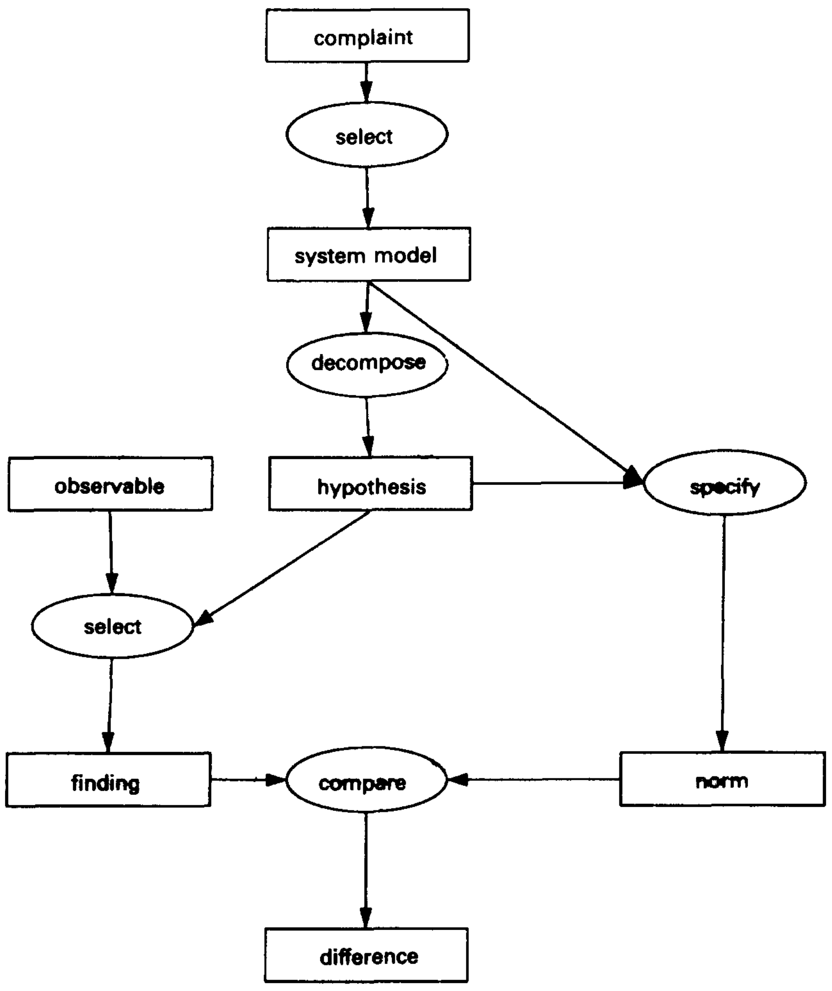


FIGURE 6. An inference structure for diagnosing faults in an audio system.

inference structure thus specifies the problem solving *competence* of the target system.

Figure 6 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:

(i) A selection of a (sub-part) of the audio system (*system model*) on the basis of a *complaint*.
(ii) A decomposition of some part of the system into a number of sub-components that play the role of *hypothesis*.
(iii) A prediction of a *norm*-value for a hypothesis. The norm is a value of a test that is consistent with the normal state of the hypothesis.

(iv) A selection of an observable, for which a value is to be obtained (the *finding*).
(v) A comparison of the observed *finding* and the predicted *norm*.

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

4.3. 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 characterized 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, 1985*a*; Gruber, 1989) have pointed out that task knowledge is an important element of expertise. The competence model of the diagnostic strategy of NEOMYCIN (Clancey, 1985*a*) is an example of what we call task knowledge. Clancey describes the sub-tasks of this strategy via meta-rules. The main difference 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 application of the task to a particular (sub-)problem results in the achievement of a goal.

Control terms
The vocabulary used. A control term is nothing more than a convenient label for a set of meta-class elements. The label represents a term used in the control of problem solving, e.g. "differential" or "focus". Each control term is defined through the specification of a mapping of this term on to sets of meta-class elements (e.g. the differential is the set of all active hypotheses).

Task structure
A decomposition into sub-tasks and a specification of the control dependencies between these sub-tasks.‡ The decomposition can involve three types of sub-tasks: (i) primitive problem solving tasks: inferences specified in the inference layer, (ii) composite problem solving tasks: a task specified in the task layer. (In principle, this could be a recursive invocation of the same task.) and (iii) transfer tasks: tasks that require interaction with an external agent, usually the user.

† 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, Akkermans & Wielinga (1991) for a more elaborate discussion on these different types of control.

‡ We agree with Steels (1990) that "control structure" is a more appropriate term for this type of structure. The term "task structure" is used here mainly for historical reasons.

The dependencies between the sub-tasks are described as a structured-English procedure such as used in conventional software engineering (DeMarco, 1978), 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**. . .".

There is interaction between the task knowledge in the model of expertise on the one hand and the model of cooperation on the other, with respect to the specification of the transfer tasks. Transfer tasks are more-or-less specified as a black box in the model of expertise. We distinguish four types of transfer tasks (for more details, see de Greef & Breuker, 1992, this issue): (i) *obtain*: the system requests a piece of information from an external agent. (The system has the initiative.), (ii) *present*: the system presents a piece of information to an external agent. (The system has the initiative.), (iii) *receive*: the system gets a piece of information from an external agent. (The external agent has the initiative.) and (iv) *provide*: the system privides 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)
        generage-hypotheses(inconsistent-sub-system→
            differential)
      UNTIL differential=∅
```

For readability purposes, the names of knowledge sources are italicized 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*). Testing is done through a kind of experimental validation: a norm value is predicted and this value is compared with what is actually observed. *Obtain*(*observable, finding*) is an example of a transfer task, that starts an iteration 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 and assumes a fixed task decomposition, the set of task structures can be represented graphically as a tree. The tree for systematic diagnosis is shown in Figure 7. Such a decomposition of a task assigned to the system is in fact a further refinement of the decomposition specified in the task model (see Section 3).

4.4. STRATEGIC KNOWLEDGE

The fourth category of knowledge is the *strategic knowledge*.† Strategic knowledge determines what goals are relevant to solve a particular problem. How each goal is achieved is determined by the task knowledge. Strategic knowledge will also have to deal with situations where the afore-mentioned knowledge categories fail to produce

† 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.

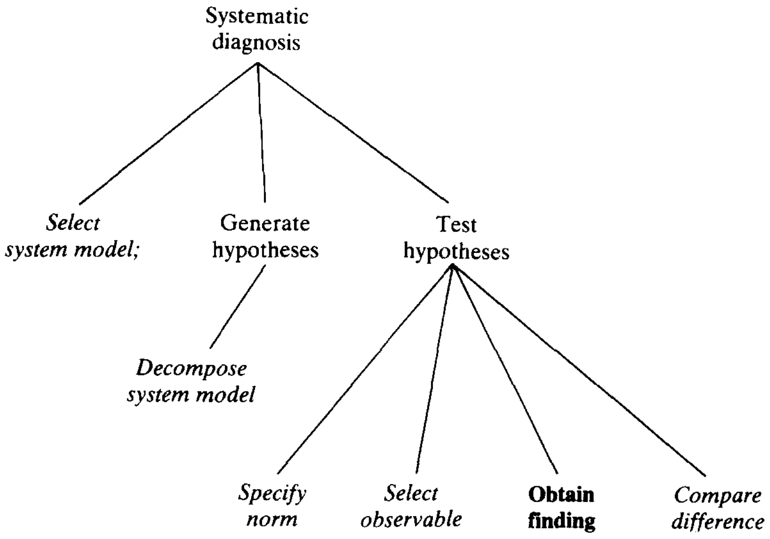


FIGURE 7. Task tree of systematic diagnosis. The leaves of such a tree are either knowledge sources or transfer tasks.

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, Elshout & Wielinga, 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 mainly a research topic.

4.5. 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 8 these four levels and their interrelations are summarized.

The four-layer framework is a structured but informal framework. This means that the specifications are sometimes not as precise as one might want them to be and thus may be interpreted in more than one way. This has led to research aimed at defining a formal framework for representing models of expertise (van Harmelen

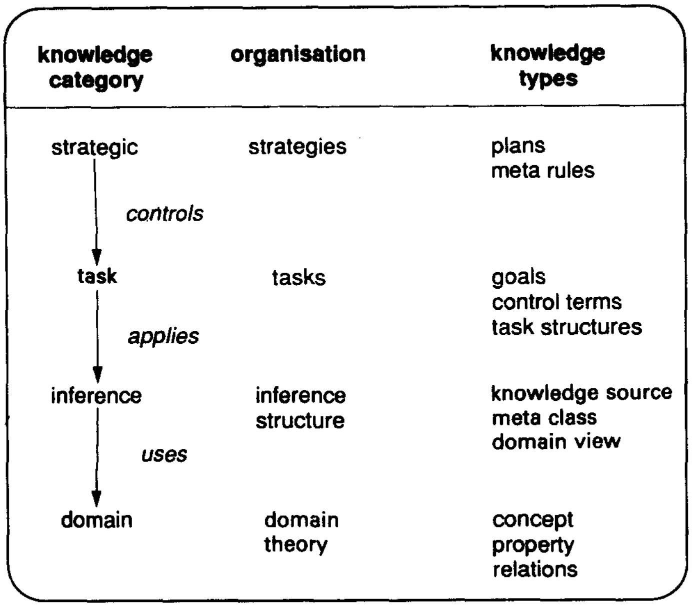


FIGURE 8. Synopsis of the KADS Four-Layer Model.

& Balder, 1992, this issue; Wetter, 1990). The price paid for a greater amount of precision in formal specifications is, however, a reduction in conceptual clarity. In our view, there is a place for both informal and formal representations in the knowledge engineering process. The use of both informal and formal model representations is a major topic of research in the KADS-II project.

The four-layer framework for knowledge modelling has been successfully used as a basis for structured acquisition and description of knowledge at an intermediate level between the expertise data obtained from experts, test books etc. 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 particular types of knowledge. This point will be discussed in more detail in Section 9.

5. Principle 3: re-usable model elements

There are several ways in which models of expertise can be used to support the knowledge acquisition process. A potentially powerful approach is to *re-use* (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 re-occur 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 predefined sets of model elements. These libraries can be of great help to the knowledge engineer. They provide the engineer with ready-made building blocks and prevent him/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, two ways of re-using elements of the model of expertise are discussed: (i) *typologies* of primitive inference actions (knowledge sources) and (ii) *interpretation models.* In principle, however, the re-usability principle holds for all models in the KBS construction process.

5.1. TYPOLOGIES OF KNOWLEDGE SOURCES

In Breuker *et al.* (1987) we have defined a tentative typology of primitive problem solving actions (knowledge sources) which has been the basis of a considerable amount of models. The typology is based on the possible operations one can perform on the epistemological primitives defined in KL-ONE (Brachman & Schmolze, 1985). This set of primitives consists of: concept, attribute (of concept), value (of attribute), instance (of concept), set (of concepts) and structure (of concepts).

In the typology of inferences we view these primitives not as data-structures but as epistemological categories. Their actual representation in a system may be quite different (e.g. in terms of logical predicates rather than KL-ONE like constructs).

Table 2 gives an overview of the typology of knowledge sources used in KADS. The inferences are grouped on the basis of the type of operation that is carried out by the knowledge source: *generate concept/instance, change concept, differentiate values/structures* and *manipulate structures*. A detailed description of the inferences mentioned in Table 2 is given in Breuker and Wielinga (1989).

Although this typology has been a useful aid in many analyses of expertise, it has a number of important limitations:

(i) The selected set in Table 2 is in a sense arbitrary. For example, we could have added other operations on sets such as *join, union* or *merge.*

TABLE 2
A typology of knowledge sources

Operation type	Knowledge source	Arguments
Generate concept/instance	instantiate	concept → instance
	classify	instance → concept
	generalize	set of instances → concept
	abstract	concept → concept
	specify	concent → concept
	select	set → concept
Change concept	assign-value	attribute → attribute-value
	compute	structure → attribute-value
Differentiating values/structures	compare	value + value → value
	match	structure + structure → structure
Structure manipulation	assemble	set of instances → structure
	decompose	structure → set of instances
	transform	structure → structure

(ii) The ontology on which the typology is based is of a very general nature and hence weak. The operations are defined more or less independent of tasks and/or domains. Often, it is difficult for the knowledge engineer to identify how an inference in a particular application task must be interpreted.

(iii) A more serious limitation is that some inferences cannot be adequately classified because they require another ontological framework. For example, operations on causal relations such as abduction and differentiation cannot be represented in a natural way.

We consider the study of more adequate taxonomies of inferences to be a major research issue. Potentially, taxonomies are very powerful aids for the knowledge engineer. In a new research project (KADS-II) we are exploring the possibility of describing taxonomies that are specific for classes of application domains such as technical diagnosis. These taxonomies will be based on a much more task-specific ontology.

It is interesting to see that, from a different angle, the "Firefighter" project (Klinker *et al.*, 1991) is aiming at similar results. An important goal of this project is to look at what they call *mechanisms* that are used in various applications, detect commonalities between these mechanisms and construct a library of mechanisms that can be re-used in other applications. These mechanisms appear to have the same grain size as the knowledge sources in KADS. The main difference is that mechanisms have a computational flavour.

5.2. INTERPRETATION MODELS

Typologies of elements of a model of expertise, such as a typology of knowledge sources, represent a first step into the direction of re-usability. A further step would be to supply *partial models* of expertise such as models without all the detailed domain knowledge filled in. Such partial models can be used by the knowledge engineer as a template for a new domain and thus support top-down knowledge acquisition. In KADS such 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* knowledge and *task* knowledge for a particular task. As these descriptions are phrased in domain-independent terminology, they are prime candidates for re-use in other domains. For example, the inference and task description of the audio domain could very well be applied 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.

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, de Hoog & Schreiber, 1988). It is also interesting because it illustrates how different tasks can apply the same set of inferences in different ways.

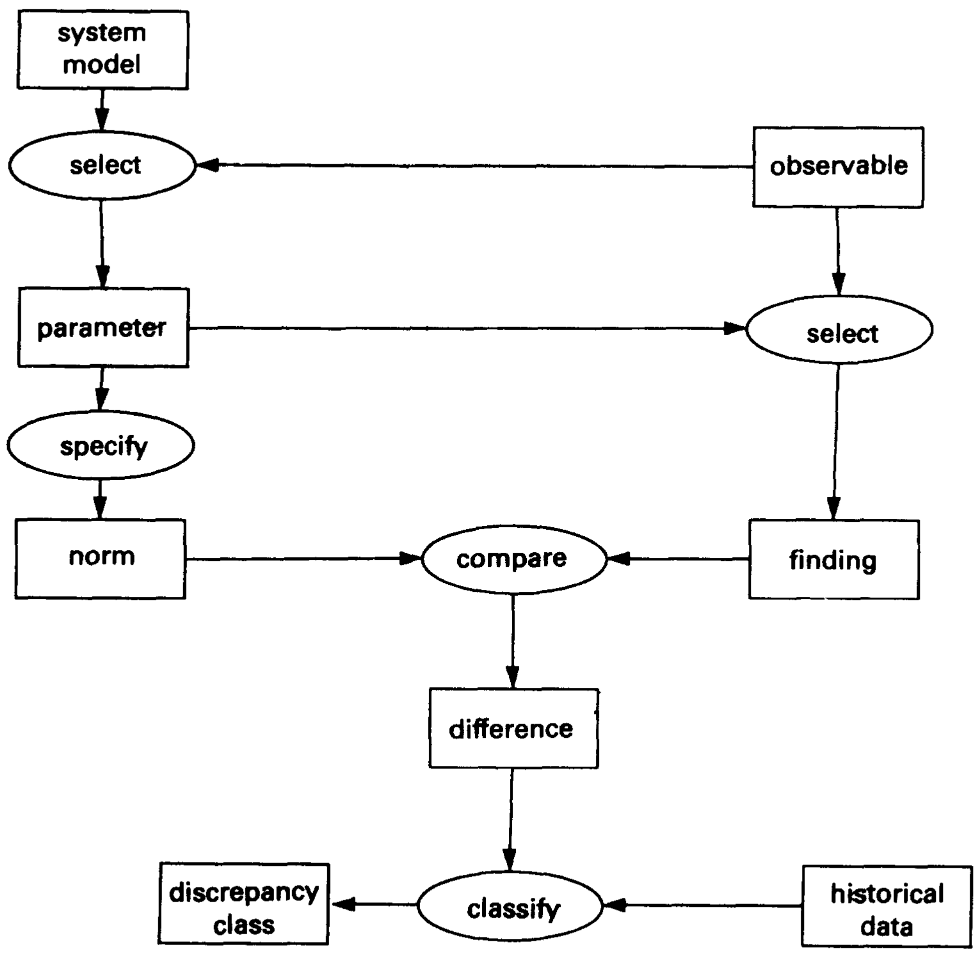


FIGURE 9. Inference structure of the interpretation model for *monitoring*.

The inference structure of the interpertation model for the monitoring task (shown in Figure 9) depicts the following inferences: (i) the selection of a system parameter, (ii) the instantiation of the normal value of the parameter (the norm), (iii) the selection of a corresponding observable, (iv) a comparison of observed and expected values, leading to a difference description and (v) 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.)

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 9.

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-parameter)
        DO FOR EACH parameter ∈ active-parameters
          specify(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 in which an external agent (a human user or another system) has the initiative (see Section 4.3). 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)
          specify(parameter→norm)
          compare(norm+finding→difference)
          classify(difference+historical-data→discrepancy)
```

Selecting an interpretation model from the library

The library of interpretation models consists of a number of models that can 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 10. The decision tree is based on a taxonomy of task types. This taxonomy is a modified and extended version of Clancey's (1985*b*) description of problem types which in turn was derived from Hayes-Roth, Waterman and Lenat (1983: p 14). The decision points in this tree concern features of the solution space, the problem space and the required domain knowledge types.

The first decision point concerns the availability of information about the structure of the system involved in a task. The term "system" refers here to the

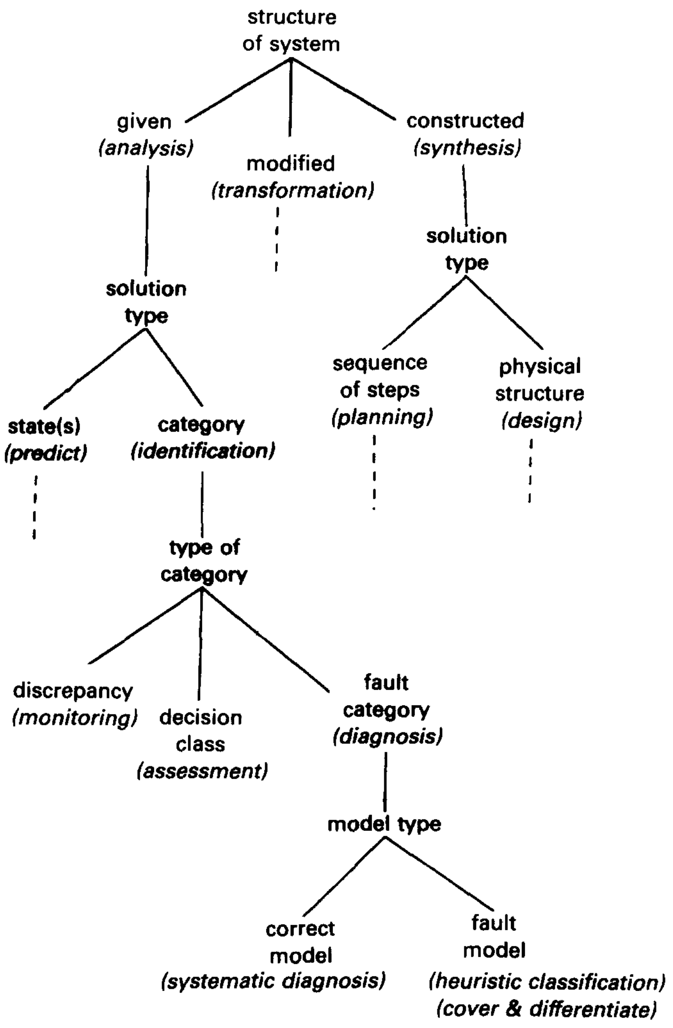

structure of system
given (analysis)
modified (transformation)
constructed (synthesis)
solution type
solution type
state(s) (predict)
category (identification)
sequence of steps (planning)
physical structure (design)
type of category
discrepancy (monitoring)
decision class (assessment)
fault category (diagnosis)
model type
correct model (systematic diagnosis)
fault model (heuristic classification) (cover & differentiate)

FIGURE 10. Partial decision tree of interpretation models.

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. 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 10. 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 between observed and expected behaviour.

It should be noted that in many real-life applications the task is a compound one: it consists of several basic tasks. For example, in the model of expertise for the audio domain, we focused only on 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).

A number of researchers have developed knowledge acquisiton tools that are based on the notion of a generic model of the problem solving task. For example, ROGET (Bennet, 1985), MOLE (Eshelman *et al.*, 1988) and BURN (Klinker *et al.*, 1991) are all systems that drive the knowledge acquisition dialogue with an expert through a strong model of the problem solving process. This model prescribes what domain knowledge is needed to build an actual expert system. In OPAL (Musen *et al.*, 1988) this approach is taken one step further. The conceptual model is OPAL is not just a model of the problem solving process (i.e. the upper three layers in the KADS framework) but also contains templates of the domain knowledge needed. As a consequence OPAL can present the expert with detailed forms that he or she can fill in with the details of an application domain. Although this approach is very powerful indeed, it has limitations in scope and applicability.

6. Principle 4: knowledge differentiation

The use of template descriptions such as interpretation models provides a powerful tool for knowledge acquisition. However, applying such a template to a particular domain will often reveal that the model does not completely fit the data on human expertise. Most interpretation models embody only a minimal set of inferences necessary for solving a problem with this method. The model needs to be further refined. This process of refinement is called *knowledge differentiation*.

Knowledge differentiation is guided by several types of characteristics of the application domain: e.g. the nature of the knolwedge in an application domain (e.g. "are causal models available?"), the constraints posed by the task environment (e.g. the required certainty of a solution) and computational constraints: it is possible to find computational techniques that realize the specified behaviour.

In the "Components of Expertise" framework (Steels, 1990) these characteristics are called "task features" and the three categories respectively *epistemic*, *pragmatic* and *computational* task features.†

6.1. TWO DIFFERENTIATIONS OF SYSTEMATIC DIAGNOSIS

The model of systematic diagnosis can be differentiated in various ways. We discuss two differentiations relevant to the audio domain.

The plain model of systematic diagnosis (of which the inference structure is shown in Figure 6) presupposes that the applicable system model is *selected* using knowledge about fixed decompositions of the system being diagnosed. However, the configuration of an audio system is usually not fixed. System elements such as a CD-player, a second tape-deck, head phones or additional speakers may or may not be present. This potential problem can be handled by replacing the simple *select* inference with a more complicated *assemble* inference (Figure 11).

† In the Components approach the task features are used for dynamic run-time task-decomposition in an actual system. In KADS, knowledge differentiation is primarily seen as a knowledge engineering activity, which could be (but does not need to be) reflected in the design of the KBS (in other words, it could result in fixed task decompositions).

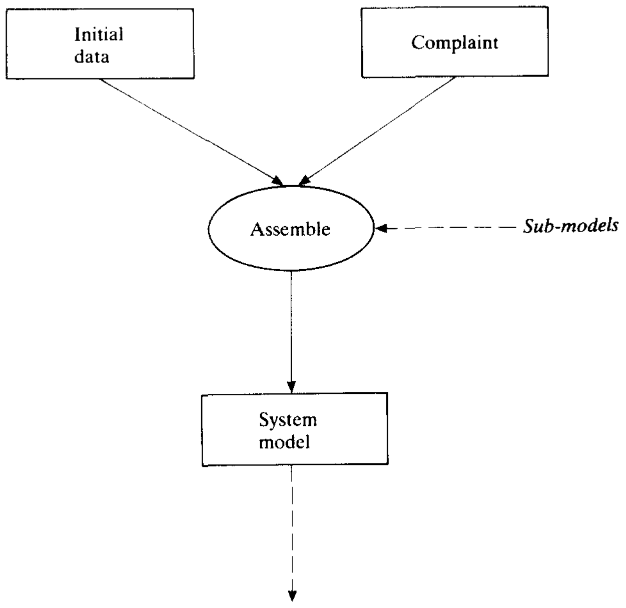


FIGURE 11. Differentiating the model of Systematic Diagnosis: *system model assembly* instead of selection.

In this assemble step additional data (*initial data* in Figure 11) about the audio system are used to construct an applicable system model. An epistemic requirement for this differentiation is that additional domain knowledge can be made available, namely: (i) a definition of potential system elements of an audio system, possible hierarchically organized (cf. the *sub-models* in Figure 11) and (ii) configuration rules for assembling an actual model from the possible system elements.

This modification of the plain inference structure of systematic diagnosis thus leads to a slightly more complex model with additional domain knowledge requirements. A second, more complicated, differentiation concerns the introduction of multiple system *views*. Often, there are various ways of decomposing a device. Each decomposition represents a different view on the system. Well-known views are functional and physical decompositions. In the audio domain, one can think of the system as, for example, an *electrical* system or as a *sound transformation* system. The faulty component can only be found if the right view is selected.

Allowing multiple views also introduces additional complexity in the model of expertise. It implies an additional decision in the inference process concerning view selection. Davis (1984) suggests that initial view selection is usually done on the basis of characteristics of the problem (the *complaint*) using domain heuristics. Figure 12 shows the additional inferences necessary for handling multiple views. In this case the epistemic requirement on additional decomposition knowledge is even stronger: for each view sub-models and configuration rules should be present in the domain theory. In addition, heursitics about how to select a view need to be made available.

Introducing multiple views also involves additional task structure complexity. If one view fails to provide a solution, another view needs to be selected and the process is repeated. We omit here the specification of this extended task structure.

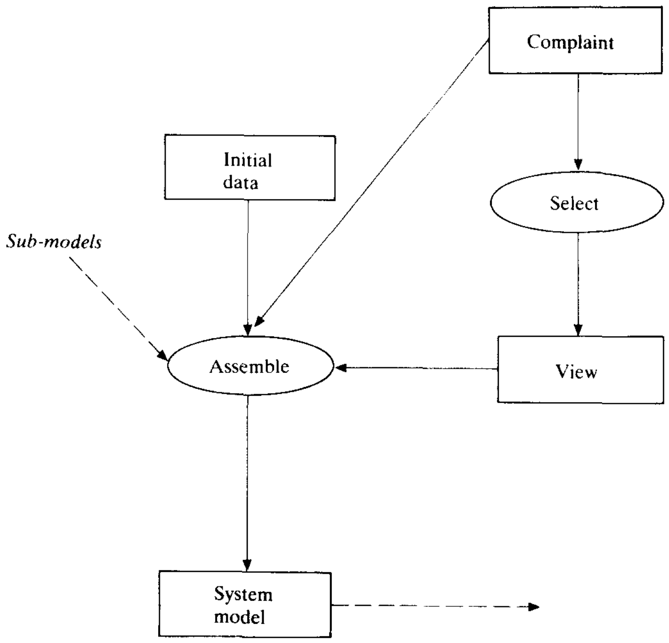


FIGURE 12. Differentiating the model of Systematic Diagnosis: introducing multiple system models representing different views.

6.2. MODEL CONSTRUCTION OPERATORS

Thus far, we have presented the process of constructing a model of expertise more-or-less as a two-step process: (i) the selection of an interpretation model from the library and (ii) the differentiation of the model on the basis of characteristics of the application domain.

More recently, we are considering a somewhat more dynamic view of the model constructuion process. This view has been influenced by the work of Patil (1988) on medical diagnosis. He shows how one can start with a simple model of diagnosis, such as generate-and-test, and start a gradual refinement process of this model on the basis of application characteristics, such as the ones discussed earlier. This approach would require a different organization of the library of template models, namely not as a flat set of models but as a set of *model construction operators*. These model construction operators can be applied to a model and result in a more complex model. The two differentiations of systematic diagnosis can be viewed as examples of such operators. Model construction operators can also be identified on a more general level. If one studies the inference structures of systematic diagnosis (Figure 6) and of monitoring (Figure 9), it becomes clear that they share a common set of related inferences, namely the process of checking the expected value of a parameter against the observed value. Figure 13 shows this set of inferences. One could view this set as a potential model construction operator.

Representing template models in the form of such model construction operators is attractive because it captures the way in which knowledge engineers actually build these models.

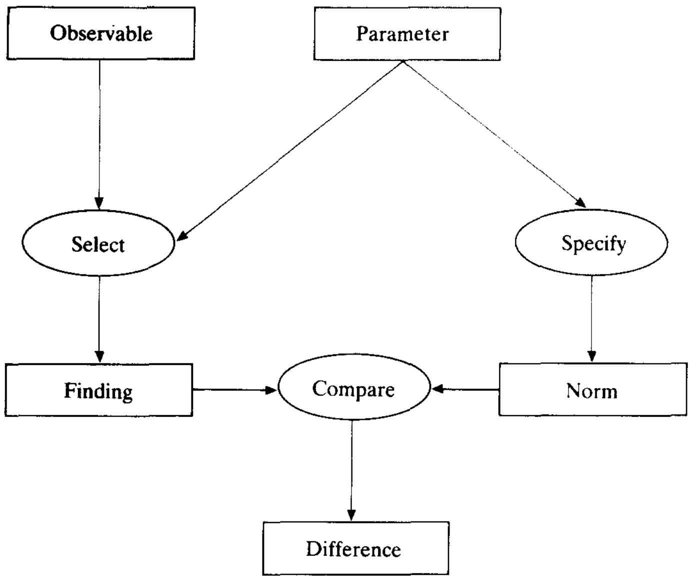


FIGURE 13. Model construction operator: checking the expected value of a parameter against the observed value.

7. Principle 5: structure-preserving design

Unlike most other approaches to knowledge modelling, the KADS models of expertise have no direct computational interpretation: they are not executable. In this section we discuss various issues that are related to the process of operationalizing the model of expertise:

(i) The trade-off between conceptual modelling and design.
(ii) The structure-preserving approach to design and its advantages.
(iii) An overview of the detailed design decisions in a structure-preserving design process.
(iv) Support for a structure-preserving design process.
(v) Computational adequacy of systems built on the basis of a non-executable knowledge-level model.

7.1. TRADE-OFF BETWEEN CONCEPTUAL MODELLING AND DESIGN

It is important to realize that there is a trade-off between (i) extending the conceptual model and (ii) a more elaborate specification during design. The decision whether to do the former or the latter depends on whether one is interested from the conceptual-model point of view in exercising explicit control on the execution of a computational technique. The borderline between conceptual model and design is thus, in a sense, governed by the level of *granularity* that is required of the conceptual model. For example, in OFFICE-PLAN (a system for allocating offices to employees, see Karbach, Linster & Voß, 1989) the actual allocation inference is modelled as one knowledge source *assign* in the conceptual model and is realized 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".

7.2. APPROACHES TO DESIGN

By definition, knowledge-level models such as the KADS conceptual models 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 the conceptual 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, 1987*a*; 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.

However, from a methodological point we strongly favour a *structure-preserving design*. With structure-preserving we mean that for the final system it should be possible to relate the elements of conceptual model to identifiable computational constructs. 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 conceptual model makes it possible to generate explanations at the level of the language of the conceptual model (Clancey & Letsinger, 1984).

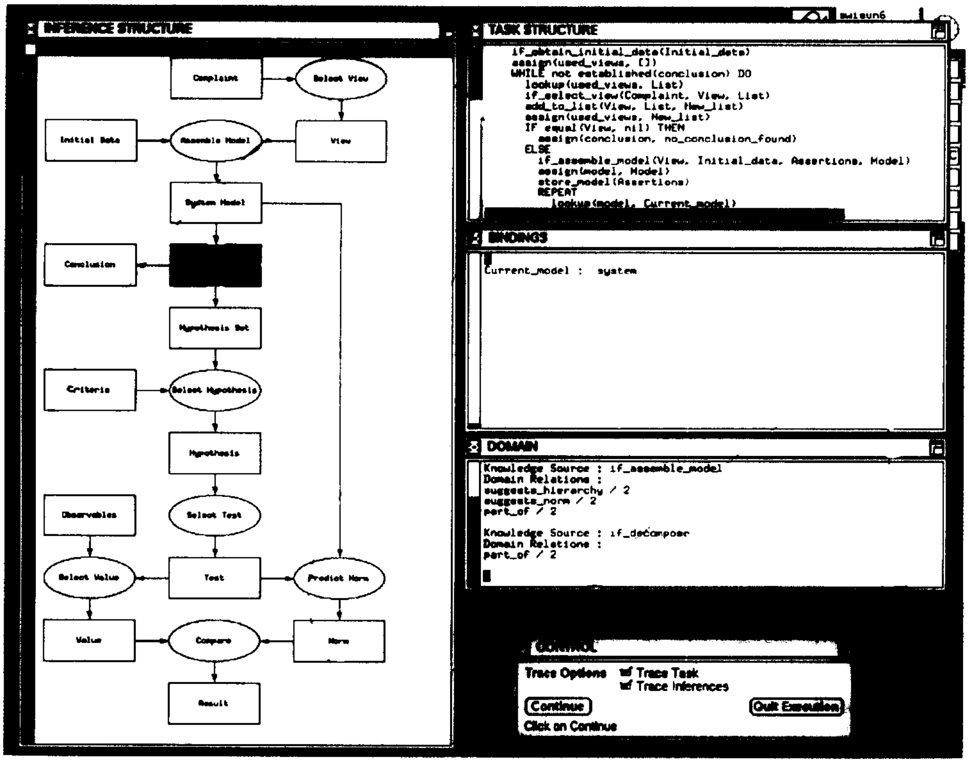
Figure 14 shows part of the interface of a prototype shell that we developed for operationalizing models of systematic diagnosis. The shell was used to implement a KBS for the audio domain (Lemmers, 1991). The interface allows the user to trace the reasoning of the KBS in the vocabulary of the conceptual model at various levels through: (i) the task that is being executed and its structure, (ii) the inference structure in which an inference is highlighted when it is being executed, (iii) the bindings of control terms and meta-classes (i.e. the current state of "working memory") and (iv) the domain knowledge that is used by inferences that are executed.

The interface is a simple example of providing a user insight into the reasoning process of the KBS.

Maintenance and debugging

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

† Similar goals with respect to explanation and maintenance are pursued in the Explainable Expert Systems (EES) approach (Neches, Swartout & Moore, 1985). In EES the structure-preserving property is guaranteed through an automatic transformation process.



```
if_obtain_initial_data(Initial_data)
assign(used_views, [])
WHILE not established(conclusion) DO
  lookup(used_views, List)
  if_select_view(Complaint, View, List)
  add_to_list(View, List, New_list)
  assign(used_views, New_list)
  IF equal(View, nil) THEN
    assign(conclusion, no_conclusion_found)
  ELSE
    if_assemble_model(View, Initial_data, Assertions, Model)
    assign(model, Model)
    store_model(Assertions)
    REPEAT
      lookup(model, Current_model)
```

```
Current_model :  system
```

```
Knowledge Source : if_assemble_model
Domain Relations :
suggests_hierarchy / 2
suggests_norm / 2
part_of / 2

Knowledge Source : if_decompose
Domain Relations :
part_of / 2
```

FIGURE 14. Prototype interface for tracing the execution of a system for systematic diagnosis in the vocabulary of the model of expertise. The inference structure is shown on the left and is a refined version of the one presented in this paper. The *decompose* knowledge source is currently being executed. The task structure, the bindings (e.g. the system model that is currently being decomposed) and the domain knowledge used by the decompose knowledge source (a part-of relation) are shown on the right. The window in the lower-right corner allows the user to trace the reasoning process at the task and/or inference level.

Knowledge acquisition tools

A related advantage is that the conceptual model 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. This approach to knowledge acquisition support allows much more focused use of tools and therefore increases the interpretability and the quality of their results. This approach is currently being pursued in the ACKNOWLEDGE project (Shadbolt & Wielinga, 1990).

7.3. DETAILED DESIGN DECISIONS

In this sub-section we discuss typical design decisions. The scope of the section is limited to decisions with respect to elements of the model of expertise. A more detailed description of these decisions can be found in Schreiber *et al.* (1987) and Schreiber *et al.* (1989*a*). Decisions concerning the design of inference knowledge are discussed first, because these decisions influence other decisions, notably the design of the domain knowledge base.

Inference knowledge
For each knowledge source a corresponding computational technique needs to be selected that can realize 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 realizing 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 synchronized 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, Soundarajan & Chandrasekaran, 1987) and logical representation and deduction (Reichgelt & van Harmelen, 1986). In Schreiber *et al.* (1989*a*) criteria for choosing a particular group of techniques are discussed. Often, knowledge engineers use, within one system, only one or two groups of techniques. In that case the chosen group of techniques takes the role of a design *paradigm*. For example, in NEOMYCIN (Clancey, 1985*a*) four (forward chaining) production system techniques are provided. Each inference applies one of these production system techniques. Applying only one type of technique, such as production systems, in one particular KBS minimizes the interaction problems with the design of other parts of the system (in particular, the domain theory, as it requires just a single representational formalization), 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 provide). Computational techniques require an access path into 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 (or 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 representations of elements of the domain-theory. In such cases the design is not fully structure-preserving. It also implies that the advantages of separating these the domain theory and the domain view (see Section 4.2) do not hold for the final system. This can mean that problems arise, particularly in the area of maintenance. For example, it could lead to multiple presence in the domain theory of essentially the same knowledge (knowledge redundancy).

Task knowledge

Given the set of tasks specified in the conceptual model (consisting of both problem solving tasks and of transfer tasks) the designer has to make two—interrelated—decisions.

(i) 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.
(ii) The choice of how to represent and update the run-time data. These data can be viewed as the "working memory" of the KBS. This working memory contains the data that are manipulated by the tasks and the inferences, e.g. the current state of the differential, the findings etc. The control terms and the meta-classes specified in the conceptual model form the basis for the representation of working memory: they often re-appear in the final system as labels for (sets of) working memory elements.

    Most existing KBSs 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 techniques, will be used more often. Note that the ue of such a technique can pose additional requirements on the output produced by computational techniques realizing primitive inferences. An example of such an additionally required output is the "justification" used in an ATMS (de Kleer, 1986).

Strategic knowledge

Most conceptual models 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" into fixed task decompositions with possibly a few strategic decision points that can be influenced by the user (cf. de Greef, Schreiber & Wielemaker, 1987, Part II, Section 3.4).

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

(i) A production system containing a set of meta-rules with states of working memory as conditions and task activations and/or changes to working memory (e.g. assumptions) as actions.
(ii) An extended blackboard technique such as the "Blackboard Control Architecture" (Hayes-Roth, 1985), where the scheduling part represents the strategic knowledge.
(iii) One can also view the strategic knowledge as a separate meta-system, the PDP system is an example of this approach (Jansweijer, Elshout & Wielinga, 1986). This approach is also currently being pursued in the REFLECT project (Reinders *et al.*, 1991).

7.4. SUPPORT FOR THE DESIGN PROCESS

The design decisions should lead to a mapping from the elements of the conceptual model on to elements of the implementation environment. In KADS we have developed a notation for supporting this structure-preserving design process through intermediate design descriptions (Schreiber *et al.*, 1989*b*). It is also possible to support the knowledge engineer in this transformation process through the development of a dedicated computational framework that provides: (i) a number of predefined representational and computational techniques, and (ii) a programming environment that more-or-less predefines how elements of the conceptual model should be mapped on to constructs in this environment (for example, by providing constructs called "knowledge source" etc.).

In such a computational framework most of the design decisions discussed earlier are hard-wired. The use of a dedicated framework minimizes the number of design decisions that have to be made for implementing the system given a conceptual model, because most of the these decisions have already been taken by the designers of the environment. Model-K (Karbach *et al.*, 1991) and ZDEST-2 (Tong, He & Yu, 1988; Karbach, Tong & Voß, 1988) are computational environments that (partially) allow such a mapping of KADS conceptual models on to an implementation. A potential problem with this type of approach is that the environment 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.

7.5. 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 applied within the constraints specified by the model. For instance, a piece of domain knowledge 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 (Schreiber *et al.*, 1991).

8. The knowledge-acquisition process

The description of the various models can be seen as the *product* of KBS construction, With respect to the *process* of KBS construction, KADS provides two ways of support: (i) a description of phases, activities and techniques for knowledge engineering and (ii) computerized support tools. Both are briefly discussed in this section.

8.1. PHASES, ACTIVITIES AND TECHNIQUES

A *phase* represents a typical stage in the knowledge engineering process. A phase is related to a number of *activities*, that are usually carried out in this phase. One particular activity can occur in more than one phase. For example, "data collection" can occur in many different phases. The activities are the central entities in the process view on knowledge engineering. An activity is a piece of work that has to be carried out by the knowledge engineer. An activity produces a result. This result constitutes either directly a part of one or more models or it represents some intermediate product, that is used by other activities. An activity applies one or more *technqiues*. For example, a "time estimation" activity can be carried out with an extrapolation technique. Life cycle models predefine particular phases, activities, techniques and products and also their interrelations. Life-cycle models for KADS have been described in Barthélemy *et al.* (1987) and Taylor *et al.* (1989). We limit the discussion here to those activities that are related to building a first model of expertise. We distinguish two phases in building such a first model of expertise: *knowledge identification* and *knowledge modelling*.

Knowledge identification is more or less a preparation phase before the actual construction of the model of expertise can begin. Relevant activities for this phase are shown in Figure 15 together with applicable products and techniques. The results include a task model and also intermediate products that are used by activities in other phases, especially the knowledge modelling phase. Example activities are glossary and lexicon construction. A glossary and a lexicon provide a way of documenting the application domain without having to be committed to any formal conceptualization.

In the knowledge modelling phase the knowledge engineer constructs a model of expertise. Figure 16 summarizes the main activities relevant for knowledge modelling. A crucial one is the selection of an interpretation model. This activity is supported through the decision tree discussed in Section 5. The model validation and model differentiation activities often make use of protocol analysis techniques. Model validation can also be supported by transformation of the model into a functional prototype. This prototype can be seen as a simulator of the problem solving aspects of the artefact. The KADS-II project is currently working on a tool to support this type of prototyping. Other activities deal with the definition of the domain conceptualization. In KADs we usually assume that in the resulting model

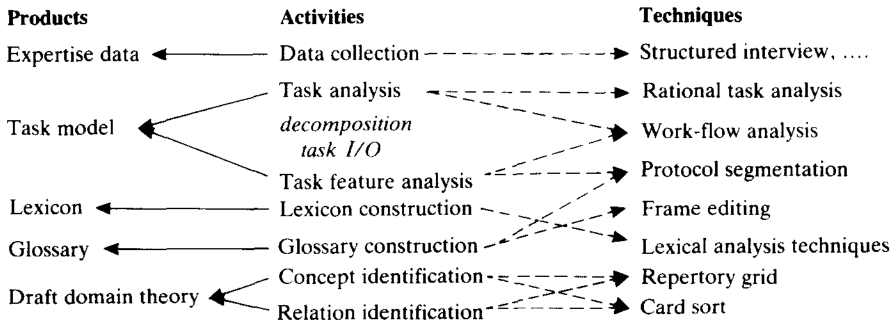


FIGURE 15. Knowledge identification activities and related products and techniques.

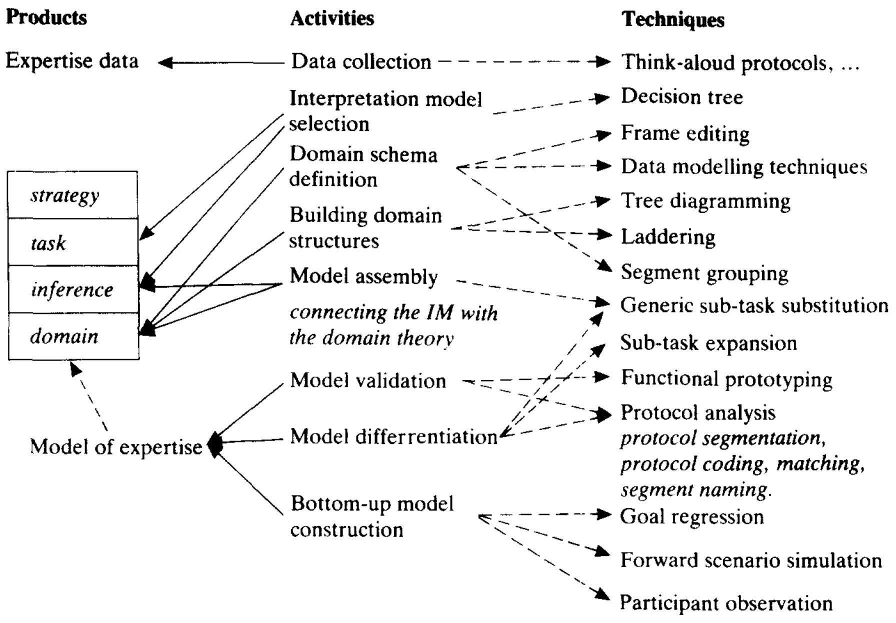


FIGURE 16. Knowledge modelling activities and related products and techniques.

the domain theory can be a partial one, but with a fully defined domain schema. Refinement and debugging of the domain theory is performed in a later phase, possibly with the use of automized techniques.

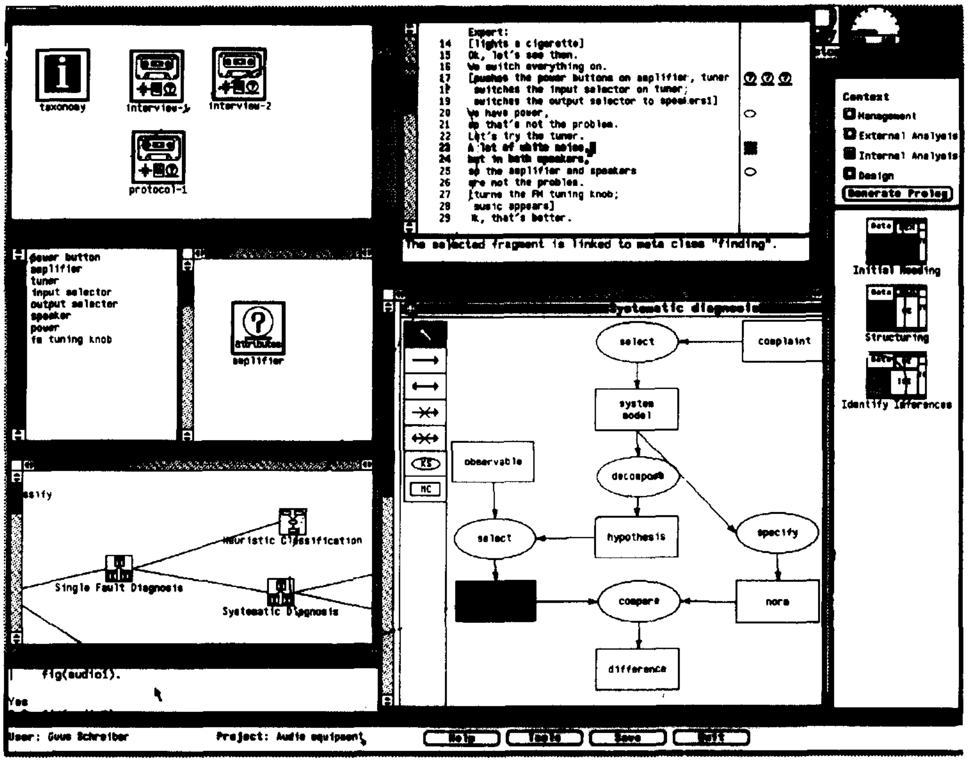
8.2. TOOLS

Within the KADS project the Shelley workbench was developed to support activities in the KBS life cycle. Shelley contains an integrated set of computerized support tools. The user of the workbench is the knowledge engineer. Example support tools in Shelley for the knowledge modelling phase are:

(i) A *domain text editor*: a tool that allows management and analysis of protocols or other texts: for example, through the creation of text fragments of a particular type. These fragments can subsequently be linked to ther objects, such as elements of the model of expertise.
(ii) A *concept editor* to create concepts and corresponding attributes.
(iii) An *interpretation model library* from which models can be selected.
(iv) An *inference stucture editor* that supports the construction of the inference layer of the model of expertise.

Figure 17 shows an example of the use of Shelley in the audio domain. The knowledge engineer has selected the interpretation model of systematic diagnosis from the library and inserted it into the inference structure editor. A think-aloud protocol is being analysed. The link of a particular fragment of the protocol to a meta-class in the inference structure editor is shown.

The Shelley workbench is described in more detail in Anjewierden, Wielemaker and Toussaint (1992, this issue).



```
      Expert:
14    [lights a cigarette]
15    Ok, let's see then.
16    We switch everything on.
17    [pushes the power buttons on amplifier, tuner
18     switches the input selector on tuner;
19     switches the output selector to speakers1]
20    We have power,
21    so that's not the problem.
22    Let's try the tuner.
23    A lot of white noise,
24    but in both speakers,
25    so the amplifier and speakers
26    are not the problem.
27    [turns the FM tuning knob;
28     music appears]
29     Ok, that's better.
```

The selected fragment is linked to meta class "finding".

```
power button
amplifier
tuner
input selector
output selector
speaker
power
fm tuning knob
```

FIGURE 17. Example session with the Shelley workbench.

9. 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 (1979) proposed five levels for describing knowledge: the linguistic, the conceptual, the epistemological, the logical and the implementational level. Brachman and also 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 Brachman's 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. Clancey (1983, 1985*b*) introduced the notion of an inference structure in the description of the model of heuristic classification (HC). In the work on NEOMYCIN (Clancey, 1985*a*) a similar type of task decomposition is found as is used 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.

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. GTs can perform quite general information-processing tasks. The assumption is that by combining a relatively small set of GTs 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 were built that provide an operationalization 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 acquisiton 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.*, 1991), 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 predefined mechanisms. As remarked in Section 5.1, 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. In this approach two steps are distinguished 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 with 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 formalizations used in Ontological Analysis are based on algebraic specification languages.

Although terminology is different, a common view appears to emerge 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.

There are also relations between KADS and conventional software engineering approaches. The introduction of multiple models was inspired by work of DeMarco (1982). As pointed out in Section 4.1, issues concerning modelling of domain knowledge are quite closely related to research in semantic database modelling. Software engineering techniques are used in KADS, e.g. a form of data-flow diagrams (for inference structures) and structured English (for task structures). Life-cycle models using a water-fall approach (Barthélemy *et al.*, 1987) and a spiral model approach (Taylor *et al.*, 1989) have been defined in KADS.

10. Experiences

The KADS approach has been (and is being) used in some 40–50 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-1 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.* 1988*b*), the integration qualitative reasoning approaches (Bredeweg & Wielinga, 1988), network management (Krickhahn *et al.*, 1988: Readdie & Innes, 1987*b*), mould configuration (Barthélemy *et al.*, 1988), mixer configuration (Wielemaker & Billault, 1988), technical diagnosis (Wright *et al.* 1988), insurance (Brunet & Toussaint, 1990) and credit card fraud detection (Land *et al.*, 1990). Other applications include re-engineering of ONCOCIN (Linster & Musen, this issue), 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, van der Wouden & Ysbrandy, 1990). The last two systems and the credit card system have been in operational use for some time.

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 find certain aspects of KADS attractive, but it is also recognized that there are many weaknesses in current KADS. The first KADS user meeting (Überreiter & Voß, 1991) in which some 40, mainly German, KADS users participated, provided a good overview of the strong and weak points of KADS. Among the strong points are:

(i) The distinction between various models, especially the distinction between the model of expertise and the design.
(ii) 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.
(iii) 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:

(i) The vocabulary in the four-layer framework for describing domain knowledge and task knowledge is not expressive enough.
(ii) The typology of knowledge sources is too general. The precise meaning of the knowledge sources is ambiguous.
(iii) The library of interpretation models is incomplete and needs serious revision. For example, coverage of synthetic tasks is marginal.
(iv) KADS does not provide enough support for operationalizing conceptual models.
(v) KADS gives you a vocabulary, but it provides little support for the modelling process.

In short, the experiences show that the KADS approach has some interesting and attractive features, but that it still needs a lot of work before it can really be considered a "comprehensive methodology". In addition, controlled validation studies are necessary to show that KADS actually provides advantages compared to other approaches. The work of Linster & Musen (1992, this issue) can be seen as a step in this direction.

11. Future developments and 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 built 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 the problem-solving methods embodied in KBSs 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 also worthwhile investigating the different types of knowledge and their relationships from a more formal point of view. Attempts are being made in this direction (see van Harmelen & Balder, 1992, this issue). 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 probelm-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 an invaluable tool for knowledge acquisition. Also, such a collection of generic models could be the basis of powerful knowledge-acquisition tools that communicate both 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

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 discussion, applications and validations of the ideas presented here.

Anjo Anjewierden, Frank van Harmelen, Marc Lister, Jacobijn Sandberg and three anonymous referees provided valuable comments on earlier versions of this paper.

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 UK), NTE NeuTech (Germany), Cap Sesa Innovation (France), and the University of Amsterdam (The Netherlands).

The production of this article was supported by the KADS-II project. This project is partially funded by the ESPRIT Progrmame of the Commission of the Euroepan 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), IBM France (France), Lloyd's Register (UK), Swedish

Institute of Computer Science (Sweden), Siemens AG (Germany), Touche Ross MC (UK), University of Amsterdam (The Netherlands) and the Free University of Brussels (Belgium).
This paper reflects the opinions of the authors and not necessarily those of the consortia.

References

ALEXANDER, J., FREILING, M., SHULMAN, S., REHFUSS, S. & MESSICK, S. (1988). Ontological analysis: an ongoing expriment. In J. BOOSE & B. GAINES, Eds. *Knowledge-Based Systems, Volume* 2: *Knowledge Acquisition Tools for Expert Systems*, pp. 25–37. London: Academic Press.

ANJEWIERDEN, A., WIELEMAKER, J. & TOUSSAINT, C. (1992). Shelley—computer-aided knowledge engineering. *Knowledge Acquisition*, **4**, 109–125.

BARTHÉLEMY, S., EDIN, G., TOUTAIN, E. & BECKER, S. (1987). *Requirements analysis in KBS development.* ESPRIT Project P1098 Deliverable D3 (task A2), Cap Sogeti Innovation.

BARTHÉLEMY, S., FROT, P. & SIMONIN, N. (1988). *Analysis document experiment F4.* ESPRIT Project P1098, Deliverable E4.1, Cap Sogeti Innovation.

BENNET, J. (1985). ROGET: A knowledge-based system for acquiring the conceptual structure of a diagnostic expert system. *Journal of Automated Reasoning*, **1**, 49–74.

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.

BOEHM, B. (1988). A spiral model of software development and enhancement. *IEEE Computer*, 61–72.

BRACHMAN, R. (1979). On the epistemological status of semantic networks. In N. FINDLER, Ed. *Associative Networks*, New York: Academic Press.

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

BREDEWEG, B. & WIELINGA, B. (1988). Integrating qualitative reasoning approaches. In *Proceedings of ECAI-88, Munich*, pp. 195–201.

BREUKER, J. & WIELINGA, B. (1987). Use of models in the interpretation of verbal data. In A. KIDD, Ed. *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 P. GUIDA & G. TASSO, Eds. *Topics in the Design of Expert Systems*, pp. 265–296. Amsterdam: North-Holland.

BRUNET, E. & TOUSSAINT, C. (1990). *A KADS application in insurance.* ESPRIT project P1098, Deliverable E9.1, Cap Sesa Innovation.

BYLANDER, T. & CHANDRASEKARAN, B. (1988). Generic tasks in knowledge-based reasoning: The 'right' level of abstraction for knowledge acquisition. In B. GAINES & J. BOOSE, Eds. *Knowledge Acquisition for Knowledge Based Systems, Volume* 1, pp. 65–77. London: Academic Press.

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. (1985*a*). Acquiring, representing and evaluating a competence model of diagnostic strategy. In CHI, GLASER, & FAR, Eds. *Contributions to the Nature of Expertise.*

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

CLANCEY, W. & LETSINGER, R. (1984). NEOMYCIN: Reconfiguring a rulebased expert system for application to teaching. In W. CLANCEY & E. SHORTLIFFE, Eds. *Readings in Medical Artificial Intelligence: the First Decade*, pp. 361–381. Reading: Addison-Wesley.

DAVIS, J. & BONNEL, R. (1990). Producing visually-based knowledge specifications for acquiring organizational knowledge. In B. WIELINGA, J. BOOSE, B. GAINES, A. SCHREIBER & M. VAN SOMEREN, Eds. *Current Trends in Knowledge Acquisition*, pp. 105–122, Amsterdam: IOS Press.

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

DAVIS, R. (1984). Diagnostic reasoning based on structure and behavior. *Artificial Intelligence*, **24**, 347–410.

DE GREEF, P. & BREUKER, J. (1985). A case study in structured knowledge acquisition. In *Proceedings of the 9th IJCAI*, p. 390–392, Los Angeles.

DE GREEF, P. & BREUKER, J. (1989). A methodology for analysing modalities of system/user cooperation for KBS. In J. BOOSE, B. GAINES & J GANASCIA, Eds. *Proceedngs of the European Knowledge Acquisition Workshop, EKAW'89*, pp. 462–473, Paris, France.

DE GREEF, P. & BREUKER, J. (1992). Analysing system-user cooperation. *Knowledge Acquisition*, **4**, 89–108.

DE GREEF, P., BREUKER, J. & DE JONG, T. (1988*a*). *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. (1988*b*). 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. (In Dutch.)

DE HOOG, R., SOMMER, K. & VOGLER, M. (1990). *Designing knowledge-based systems; a study of organisational aspects.* Technical Report Report W17, ISBN 90 346 2400 5, Dutch Organisation for Technological Aspects Research NOTA. (In Dutch.)

DE JONG, T., DE HOOG, R., & SCHREIBER, G. (1988). Knowledge acquisition for an integrated project management system. *Information Processing and Management*, **24**(6), 681–691.

DE KLEER, J. (1986). An assumption-based TMS. *Artificial Intelligence*, **28**, 127–162.

DEMARCO, T. (1978). *Structured Analysis and System Specification.* New York, Yourdon Press.

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

DIAPER, D., Ed. (1989). *Knowledge Elicitation: principles, techniques and applications.* Series in Expert Systems. Chichester: Ellis Horwood Ltd.

ESHELMAN, L., EHRET, D., MCDERMOTT, J. & TAN, M. (1988). MOLE: a tenaceious knowledge acquisition tool. In J. BOOSE & B. GAINES, Eds. *Knowledge Based Systems, Volume 2: Knowledge Acquisition Tools for Expert Systems*, pp. 95–108. London: Academic Press.

GENESERETH, M. & NILSSON, N. (1987). *Logical Foundations of Artificial Intelligence.* Los Altos, California: Morgan Kaufmann.

GOEL, A., SOUNDARAJAN, N. & CHANDRASEKARAN, B. (1987). Complexity in classificatory reasoning. In *AAAI-87*, pp. 421–425.

GRUBER, T. (1989). The acquisition of strategic knowledge. In *Perspectives in Artificial Intelligence, Volume 4.* San Diego: Academic Press.

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.

HAYES-ROTH, F., WATERMAN, D. & LENAT, D. (1983). *Building Expert Systems.* New York: Addison-Wesley.

HAYWARD, S. (1987). How to build knowledge systems; techniques, tools, and case studies.

In *Proceedings of 4th annual ESPRIT conference*, pp. 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 database 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. (1986). The expertise of novice problem solvers. In *Proceedings ECAI'86*, Brigthon.

JANSWEIJER, W., ELSHOUT, J. & WIELINGA, B. (1989). On the multiplicity of learning to solve problems. In H. MANDL, E. DE CORTE, N. BENNETT, & H. FRIEDRICH, Eds. *Learning and Instruction: European research in an international context*, pp. 127–145. Oxford, UK: Pergamon Press.

KARBACH, W., LINSTER, M. & VOß, A. (1989). OFFICE-PLAN: Tackling the synthesis frontier. In D. METZING, Ed. *GWAI-89: 13th German Workshop on Artificial Intelligence, Informatik Fachberichte* **216**, pp. 379–387. Berlin: Springer Verlag.

KARBACH, W., LINSTER, M. & VOß, A. (1990). Model-based approaches: One label—one idea? In B. WIELINGA, J. BOOSE, B. GAINES, G. SCHREIBER, & M. VAN SOMEREN, M. Eds. *Current Trends in Knowledge Acquisition*, pp. 173–189. Amsterdam: IOS Press.

KARBACH, W., TONG, X. & VOß, A. (1988). Filling 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*, pp. 501–512.

KLINKER, G., BHOLA, C., DALLEMAGNE, G., MARQUES, D. & MCDERMOTT, J. (1991). Usable and reusable programming constructs. *Knowledge Acquisition*, **3**, 117–136.

KRICKHAHN, R., NOBIS, R., MAHLMANN, A. & SCHACHTER, M. (1988). Applying the KADS methodology to develop a knowledge-based system. In *Proceedings ECAI'88, Munich*, pp. 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.

LEMMERS, M. (1991). *A shell for systematic diagnosis: structure-preserving design of a KBS*. Master's thesis, University of Amsterdam, Social Science Informatics.

LINSTER, M. & MUSEN, M. (1992). Use of KADS to create a conceptual model of the ONCOCIN task. *Knowledge Acquisition*. **4**, 55–87.

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 S. MARCUS, Ed. *Automating Knowledge Acquisition for Expert Systems*, pp. 225–255. The Netherlands: Kluwer Academic Publishers.

MEYER, M. & BOOKER, J. (1991). *Eliciting and Analyzing Expert Judgement: A Practical Guide, Volume 5 of Knowledge-Based Systems*. London: Academic Press.

MORIK, K. (1989). Sloppy modelling. In K. MORIK, Ed. *Knowledge Representation and Organisation in Machine Learning*. Berlin: Springer Verlag.

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

MUSEN, M., FAGAN, L., COMBS, D. & SHORTLIFFE, E. (1988). Use of a domain model to drive an interactive knowledge editing tool. In J. BOOSE & B. GAINES, Eds. *Knowledge-Based Systems, Volume 2: Knowledge Acquisition Tools for Expert Systems*, pp. 257–273. London. Academic Press.

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 Transactions Software Engineering*, **11**, 337–1351.

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

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

PATIL, R. (1988). Artificial intelligence techniques for diagnostic reasoning in medicine. In H. Shobe, & AAAI, Eds., *Exploring Artificial Intelligence Survey Talks from the National Conferences on Artificial Intelligence*, pp. 347–379. San Mateo, California: Morgan Kaufmann.

POPLE, H. (1982). Heuristic methods for imposing structure on ill-structured problems: The structuring in medical diagnosis. In P. SZOLOVITS, Ed., *Artificial Intelligence in Medicine*, pp. 119–190 Boulder CO: Westview Press.

READDIE, M. & INNES, N. (1987*a*). *Network management: KBS design document*. ESPRIT Project P1098, Deliverable E3.2a, SciCon Ltd. (UK).

READDIE, M. & INNES, N. (1987*b*). *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 representation languages and control regimes for expert systems. *Knowledge Engineering Review*, **1**, 2–17.

REINDERS, M., VINKHUYZEN, E., VOß, A., AKKERMANS, H., BALDER, J., BARTSCH-SPORL, B., BREDEWEG, B., DROUVEN, U., VAN HARMELEN, F., KARBACH, W., KARSSEN, Z., SCHREIBER, G. & WIELINGA, B. (1991). A conceptual modelling framework for knowledge-level reflection. *AI Communications*, **4**.

ROTH, E. M. & WOODS, D. (1989). Cognitive task analysis: An approach to knowledge acquisition for intelligent system design. In P. GUIDA & G. TASSO, Eds. *Topics in Expert System Design*, pp. 233–264, Amsterdam: North-Holland.

SCHACHTER, M. & WERMSER, D. (1988). A sales assistant for chemical measurement equipment. In *Proceedings ECAI'88, Munich*, pp. 191–193, London: Pitman.

SCHREIBER, G., AKKERMANS, H. & WIELINGA, B. (1991). On problems with the knowledge level perspective. In L. STEELS & B. SMITH, Eds. *AISB91: Artificial Intelligence and Simulation of Behaviour*, pp. 208–221, London: Springer Verlag. Also in J. BOOSE & B. GAINES, Eds. *Proceedings Ban, '90 Knowledge Acquisition Workshop*, pp. 30-1–30-14. University of Calgary: SRDG Publications.

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. (1989*a*). *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). Modeling in KBS development. In *Proceedings of the 2nd European Knowledge Acquisition Workshop*, Bonn, GMD-Studien **143**, pp. 7.1–7.15, St. Augustin. GMD. Also in: *Proceedings of the 8th Expert Systems Workshop*, Avignon, 1988.

SCHREIBER, G., WIELINGA, B., HESKETH, P. & LEWIS, A. (1989*b*). *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 M. VAN SOMEREN, & A. SCHREIBER, Eds. *Proceedings First Dutch AI Conference NAIC'88*, pp. 58–71, University of Amsterdam. Department of Social Science Informatics. (In Dutch.)

SHADBOLT, N. & WIELINGA, B. (1990). Knowledge based knowledge acquisition: the next generation of support tools. In B. WIELINGA, J. BOOSE, B. GAINES, G. SCHREIBER & M. VAN SOMEREN, Eds. *Current Trends in Knowledge Acquisition*, pp. 313–338, Amsterdam: IOS Press.

SHORTLIFFE, E., SCOTT, A., BISCHOFF, M., CAMBELL, A., VAN MELLE, W. & JACOBS, C. (1981). ONCOCIN: An expert system for oncology protocol management. In *IJCAI-81*, pp. 876–881.

STEELS, L. (1990). Components of expertise. *AI Magazine*, Summer issue. Also as: AI Memo 88–16, AI Lab, Free University of Brussels.

TAYLOR, R., PORTER, D., HICKMAN, F., STRENG, K.-H., TANSLEY, S. & DORBES, G. (1989). System evolution—principles and methods (the life-cycle model). ESPRIT Project P1098, Deliverable Task G9, Touche Ross.

TONG, X., HE, Z. & YU, R. (1988). A survey of the expert system tool ZDEST-2. In *Proceedings ECAI'88, Munich,* pp. 113–118, London: Pitman.

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

VAN DER MOLEN, R. & KRUIZINGA, E. (1990). *OKS GAK: a feasibility study.* Master's thesis, University of Amsterdam, Department of Social Science Informatics. (In Dutch.)

VAN DER SPEK, R., VAN DER WOUDEN, H., & YSBRANDY, C. (1990). The paint advisor. *Expert Systems,* **7**(4), 190–198.

VAN HARMELEN, F. & BALDER, J. (1992). $(ML)^2$: A formal language for KADS models of expertise. *Knowledge Acquisition,* **4**, 127–161. Also as, Technical Report ESPRIT Project P5248 KADS-II/T1.2/TR/ECN/006/1.0, Netherlands Energy Research Centre ECN.

VANWELKENHUYSEN, J. & RADEMAKERS, P. (1990). Mapping knowledge-level analysis on to a computational framework. In L. AIELLO, Ed. *Proceedings ECAI'90, Stockholm,* pp. 681–686, London: Pitman.

VOß, A., KARBACH, W., DROUVEN, U. & LROEK, D. (1990). Competence assessment in configuration tasks. In L. Aiello, Ed. *Proceedings of the 9th European Conference on AI, ECAI'90,* pp. 676–681, London: Pitman.

WETTER, T. (1990). First-order logic foundation of the KADS conceptual model. In B. WIELINGA, J. BOOSE, B. GAINES, G. SCHREIBER & M. VAN SOMEREN, Eds. *Current Trends in Knowledge Acquisition,* pp. 356-375. Amsterdam: IOS Press.

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 G. VAN DER VEER & G. MULDER, Eds. *Human-Computer Interaction: Psychonomics Aspects,* pp. 290–297. Berlin: Springer-Verlag.

WIELINGA, B. & BREUKER, J. (1984). Interpretation of verbal data for knowledge acquisition. In T. O'SHEA, Ed. *Advances in Artificial Intelligence,* pp. 41–50, Amsterdam: ECAI, Elsevier Science publishers.

WIELINGA, B. & BREUKER, J. (1986). Models of expertise. In *Proceedings ECAI'81,* pp. 306–318.

WINKELS, R. & ACHTHOVEN, W. (1989). Methodology and modularity in ITS design. In *Artificial Intelligence and Education,* pp. 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.