Applying KADS to KADS: knowledge based guidance for knowledge engineering

John K.C. Kingston

AIAI-TR-158

January 1995

This paper was published in the journal "Expert Systems: The International Journal of Knowledge Engineering", 12, 1, February 1995

Artificial Intelligence Applications Institute
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
United Kingdom

Abstract

The KADS methodology [Schreiber *et al*, 1993] [Tansley & Hayball, 1993] and its successor, CommonKADS [Wielinga *et al*, 1992]) have proved to be very useful approaches for modelling the various transformations involved between eliciting knowledge from an expert and encoding this knowledge in a computer program. These transformations are represented in a series of models. While it is widely agreed that these methods are excellent approaches from a theoretical viewpoint, the documentation provided concentrates on defining what models should be produced, with only general guidance on how the models should be produced. This has the advantage of making KADS and CommonKADS widely applicable, but it also means that considerable training and experience is required to become proficient in them.

This paper reviews three projects, which investigated the feasibility of producing specific guidance for certain decisions which are required when using KADS or CommonKADS to develop a knowledge based system. Guidance was produced for the identification of the generic task addressed by a knowledge based system; for the selection of appropriate AI techniques for implementing the analysed knowledge; and for selecting a suitable tool for implementing the system. Each set of guidance was encoded in its own knowledge based system, which was itself developed with the assistance of KADS or CommonKADS. These projects therefore both studied and applied KADS and CommonKADS in order to produce knowledge based guidance for knowledge engineers.

The projects showed that it was feasible to produce heuristic guidance which could be understood, applied, and occasionally overridden by knowledge engineers. The guidance provides reasonably experienced knowledge engineers with a framework for making the key decisions required by CommonKADS, in the same way that CommonKADS provides knowledge engineers with a framework for representing knowledge. The projects also produced some new insights about CommonKADS domain modelling and about the process of task identification.

1 Introduction

The KADS methodology [Schreiber *et al*, 1993] [Tansley & Hayball, 1993] and its successor, CommonKADS [Wielinga *et al*, 1992]) are collections of structured methods for building knowledge based systems, analagous to methods such as SSADM for software engineering. The development of these methods was funded by the European Community's ESPRIT programme between 1983 and 1994. KADS and CommonKADS view the construction of KBS as a modelling activity, and so these methods require a number of models to be constructed which represent different views on problem solving behaviour, in its organisational and application context. CommonKADS recommends the construction of six models:

- a model of the organisational function and structure;
- a model of the tasks required to perform a particular operation;
- a model of the capabilities required of the agents who perform that operation;
- a model of the communication required between agents during the operation;
- a model of the expertise required to perform the operation, which is divided into three sub-levels:
  - models of declarative knowledge about the domain;
  - models of the inference processes required during problem solving;
  - an ordering of the inference processes.
- a model of the design of a KBS to perform all or part of this operation.

For more details on these models, see [deHoog *et al*, 1993].

Experience has shown that the models recommended by KADS and CommonKADS provide an excellent basis for representing the various transformations required between eliciting knowledge from an expert and encoding it in a computer program. In addition, KADS and CommonKADS provide various libraries of generic models which have proved very useful to knowledge engineers (see [Kingston, 1993], for example). However, experience has also shown that the task of developing these models is non-trivial. There are a number of decisions to be taken at each stage in the modelling process, some of which have a major impact on the models, or on the implemented system. These decisions include:

1. Deciding the approach which should be taken to modelling: should models be produced bottom-up from acquired knowledge, top-down by instantiating generic problem-solving methods which CommonKADS provides, or by an intermediate approach which selects a generic model and then modifies it in the light of domain knowledge;
2. Selecting models from a library. The library of generic inference structures is indexed according to the type of task which is being tackled: so the knowledge engineer must determine the most appropriate task type for the current task;
3. Deciding whether the design should preserve the structure of the expertise model;
4. Deciding which knowledge representations and inference techniques should be used within the design;

5. Deciding on the most appropriate tool for implementing this knowledge based system.

Much of the CommonKADS documentation concentrates on defining what should be done in order to produce models, while only specifying how it should be done in general terms. This has the advantage that it specifies the content of models without enforcing an approach on practitioners, thus making CommonKADS widely applicable and compatible with many different approaches to knowledge engineering; however, by the same token, it requires knowledge engineers to be fairly experienced at making good knowledge engineering decisions before they can make full use of KADS or CommonKADS. The CommonKADS project has provided guidance on making some of the decisions outlined above (e.g. guidance on top-down/bottom-up approaches to model construction [Wielinga, 1993] and model instantiation [Löckenhoff & Valente, 1993]); however, many of the organisations which have recognised the value of KADS and CommonKADS have had to obtain training and accept a long learning curve for each of their staff who wants to become proficient in the KADS approach.

An alternative approach to providing extensive training and experience for all staff would be to provide specific guidance on developing KADS models, thus providing in-house KADS guidance based on the company's own knowledge engineering experiences. The purpose of this paper is to review three projects which investigated the feasibility of producing guidance for some of the important decisions listed above. The tasks for which guidance was produced were:

- identifying the task type;
- selecting appropriate AI techniques at the design stage[1];
- choosing a suitable implementation tool.

These projects were all performed by students in the Department of Artificial Intelligence, University of Edinburgh as part of an M.Sc course in Intelligent Knowledge Based Systems. The students were supervised by staff from the Department of Artificial Intelligence, and by members of the Knowledge Engineering Methods group at AIAI; the supervisors also acted as experts from whom knowledge was elicited.

It is a requirement that all students on Edinburgh's M.Sc course produce functioning software as part of their project, which meant that the students were required to acquire knowledge, analyse the knowledge, and to implement this knowledge in a knowledge-based system. It therefore seemed sensible for the students to use KADS to help them in this process, since practical experience of using

[1]Guidelines on whether a design should preserve the structure of an expertise model are currently being considered.

KADS ought to help them in producing useful guidance. The main body of this paper therefore shows how KADS was used to model the tasks involved in making KADS-related decisions in these three projects.

2 Identifying task types

The first project looked at the task of identifying the most appropriate task type for a particular problem [Krueger, 1992].

2.1 The task

The student who took on this project was set the task of developing a technique for distinguishing and classifying different expert tasks, with a focus on the taxonomy of expert tasks developed by the KADS methodology (see Figure 1). This taxonomy defines the contents of the library of generic inference structures; there is a generic inference structure associated with (almost) every leaf node in the taxonomy. The selection of a suitable generic inference structure therefore boils down to the identification of the most appropriate task type from this taxonomy.

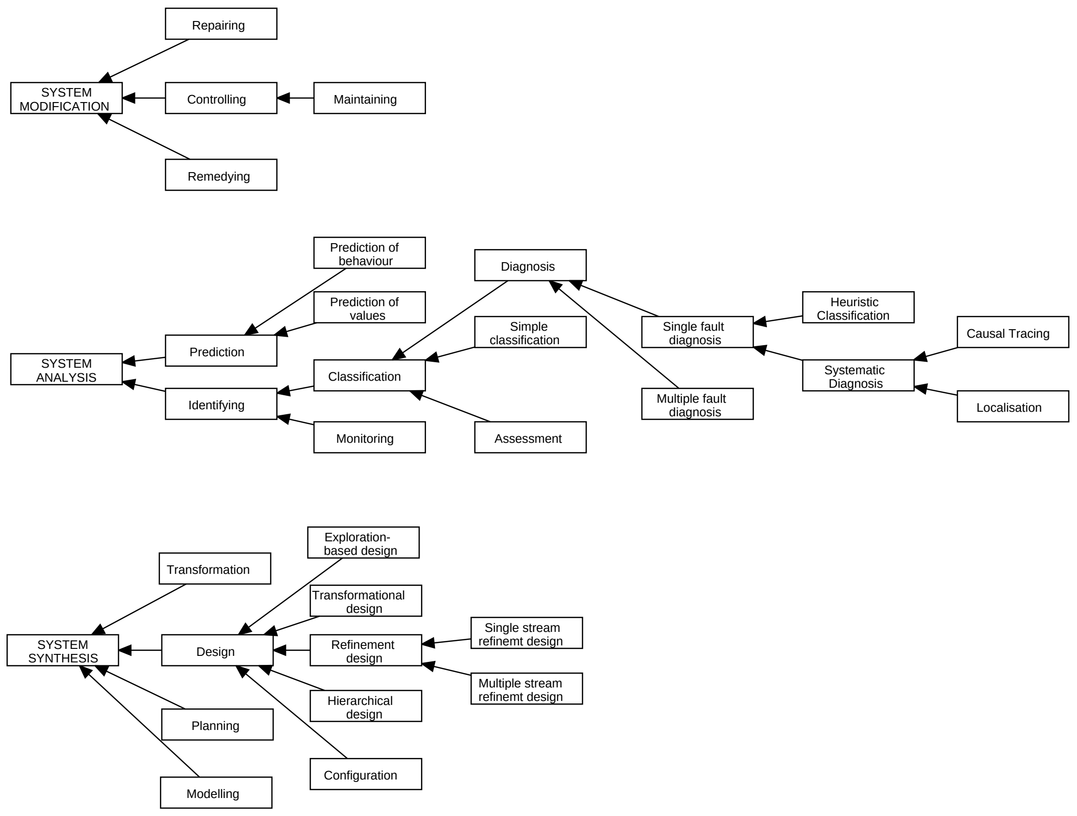

Figure 1: Taxonomy of task types

2.2 The project

Given that the student has been presented with a problem to solve using knowledge-based techniques, the first stage of KADS analysis is to identify the type of task which is carried out in order to select a generic inference structure. Fortunately, the obvious "chicken and egg" situation which arose here was circumvented by reading an early KADS report [Breuker., 1987], which suggests that the appropriate task type here is *assessment* – that is, assessing how well each of the task types matches the task under consideration, and then selecting the one which matches most closely. The student therefore designed and implemented a KBS which used an assessment approach to the identification of task types.

KADS' generic inference structure for assessment tasks (Figure 2) recommends that assessment is carried out by *abstracting* (i.e. generalising) key features of a particular problem *case description*, and *specifying* the preferred value of these features from a *system model* which represents the "ideal world". The features of the case are then *matched* against the preferred features to determine the degree of acceptability of the case. For this task, the "problem case" is an expert task, and the "ideal world" is (one of) a set of generic inference structures. The generic inference structure for assessment tasks was therefore adapted and instantiated in the ways described below: the resulting inference structure is shown in Figure 3.

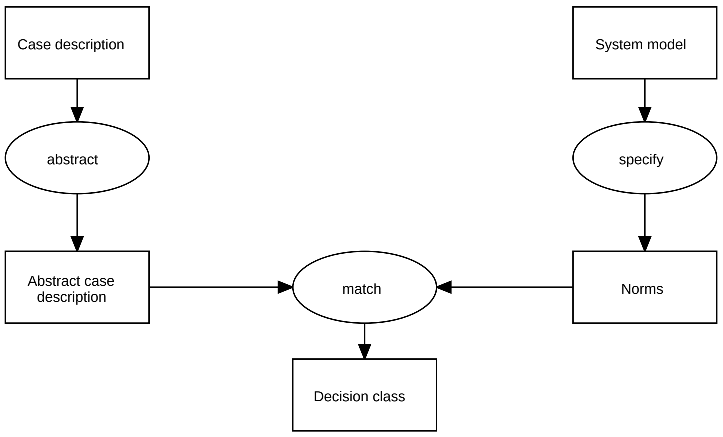

Figure 2: Generic inference structure for assessment tasks

In order to instantiate this generic structure to problem in hand, the student was required to make some alterations to the generic inference structure:

- The key features of an expert task were obtained by asking the user, rather than by abstracting from a detailed description. The abstraction inference was therefore replaced by an *obtain* step[2];

[2]In CommonKADS, *obtain* is considerd to be a *transfer task* rather than an inference step. Transfer tasks are represented by rounded rectangles.

- The set of inference structures goes through a two-stage specification: the first stage determines the features which must be asked of the user, and the second stage specifies parameters of these features which can be compared against the user's replies;
- A "feedback" from the differences discovered to the set of inference structures under consideration is explicitly recorded in the problem-specific inference structure.

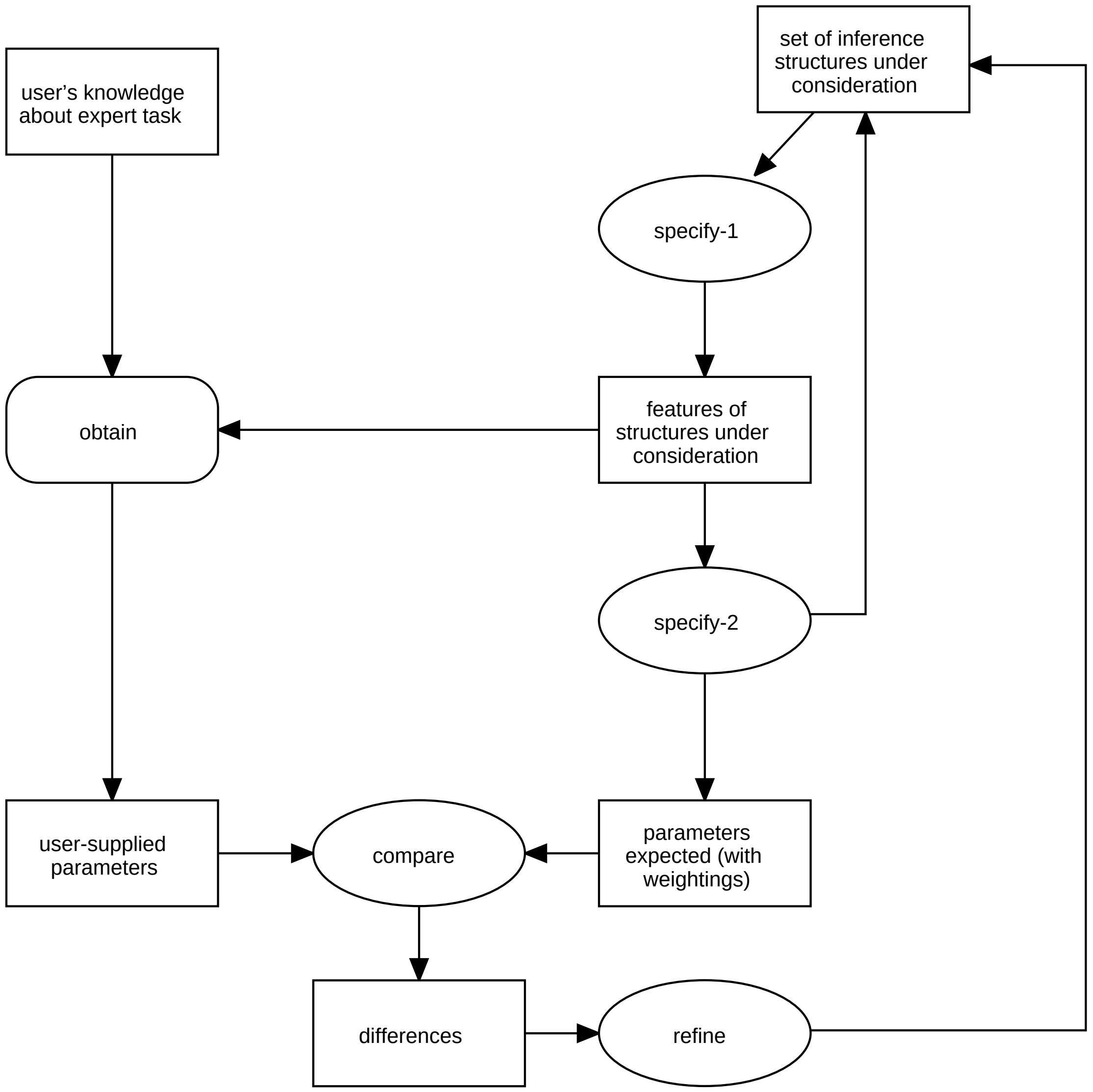
user's knowledge about expert task
set of inference structures under consideration
specify-1
obtain
features of structures under consideration
specify-2
user-supplied parameters
compare
parameters expected (with weightings)
differences
refine

Figure 3: Instantiated inference structure for assessing inference structures

The resulting system, known as SEXTANT,[3] asks users to identify inputs of

[3]**S**election of **EX**pert **TA**sks by **N**ature of the **T**ask

a task, outputs of a task, and knowledge roles which exist in the problem solving process; it then matches this information against the corresponding attributes of each generic inference structure in order to attach a likelihood weighting to each inference structure. As weightings increase or decrease, some inference structures are removed from consideration until only one (or a few) are left. The remaining inference structure(s) are then recommended to the user.

As the project progressed, it became obvious that an alternative technique for identifying task types could be developed, based on the taxonomy of task types shown in Figure 1. By starting at the topmost node in the taxonomy, it is possible to traverse the taxonomy by asking a single question at each node to decide which is the most appropriate subcategory for the current problem. For example, at the topmost level, the following question could be asked:

- Does the task involve
    1. Establishing unknown properties or behaviour of an object within the domain?
    2. Composing a new structural description of a possible object within the domain?
    3. A combination of the above?

If the first answer is chosen, then the most appropriate subcategory is *System Analysis* tasks; if the second, then *System Synthesis*; if the last, then *System Modification* is the most appropriate task type. By defining suitable questions for each non-leaf node in the taxonomy, it becomes possible to identify task types simply by answering a series of questions. Effectively, the taxonomy is transformed into a *decision tree*, and the identification of task types becomes a comparatively simple classification task.

The final version of the SEXTANT system incorporates both the "decision tree" approach and the "assessment" approach. Novice users of KADS can progress through the decision tree; however, if they are unable to answer questions in the decision tree, the assessment approach takes over, using the remaining candidates from the decision tree as the set of inference structures on which assessment is performed. Experienced users of KADS should be able to specify a relatively small set of task types at the outset, and so will only need to use the assessment part of the SEXTANT system in order to support them in their final decision. The use of SEXTANT also forces knowledge engineers to consider the nature and the dependencies of each inference step carefully, which should provide assistance in the task of instantiating the chosen generic inference structure to the actual task being performed.

SEXTANT was implemented in CLIPS 5.0, and is therefore capable of running on a range of hardware.

2.3 Results

This project demonstrated that it was feasible to provide guidance on task selection, using either an assessment-based approach or a relatively simple decision tree approach. The creation of appropriate questions for the decision tree required considerable thought about the nature of the choices being made at each choice point. The questions which were generated were reasonably consistent in their format, which suggests that the tasks in the library form a coherent set. It is not clear that the tasks form a complete set of all possible expert tasks, however, and even those tasks which are in the library do not always have an associated generic inference structure. It is likely that more work is needed on these tasks types – particularly system synthesis and system modification tasks – in order to produce a complete set of knowledge based tasks (cf. [Tansley & Hayball, 1993] [Kingston, 1994]).

Since this project was performed, the CommonKADS project has redefined generic inference structures, by simplifying the basic structures in the library and providing extensive guidance on configuring the generic structures to the requirements and features of a particular task [Löckenhoff & Valente, 1993]. This alteration has greatly increased the scope for defining many expert tasks, as variations of a smaller number of "basic tasks". It has also reduced the utility of the "assessment" approach in SEXTANT, because the differences between the simplified inference structures in the library now require less detailed analysis, and because the configuration process requires users to think deeply about the nature of the inferences performed in the task. However, SEXTANT's "decision tree" approach is still useful; indeed, it could usefully be extended to incorporate guidance on configuring inference structures, since this guidance consists of a set of questions which are used to recommend particular alterations to a basic inference structure. These "configuring" questions could therefore be considered to be extensions to the lowest levels of the decision tree.

The "decision tree" component of SEXTANT has been used on other KADS-related projects, including the projects described later in this paper.

3 Selecting appropriate AI techniques

The second project was intended to help knowledge engineers select appropriate knowledge representation and inference techniques when constructing a KBS design [MacNee, 1992].

3.1 The task

Once a task type has been identified, the acquired knowledge can be analysed by building the KADS *expertise model.* This consists of:

- an inference structure, configured and instantiated to represent the reasoning processes which take place in the task under consideration;
- a set of domain models, identifying key concepts in the domain and the relationships between them;
- a task structure, enforcing an ordering on the inference steps.[4]

The resulting expertise model must then be transformed into a program specification; this is the task of the KADS *design* phase.

The approach recommended for the CommonKADS design phase prescribes a 3-stage design process, and includes suggested approaches for modelling the first two phases [van de Velde *et al*, 1993]:

- **Application design** typically consists of a conceptual decomposition of the expertise model into a number of functional units and/or data objects;
- **Architecture design** requires decisions on whether to use rules, objects, or other representational techniques, for different parts of the application design;
- **Platform design** matches these chosen techniques with the facilities and environment offered by the chosen programming tool, with consequent alterations to the architecture design and/or the programming tool.

This approach has been used successfully on some projects, and appears to represent a sufficiently detailed breakdown of the design process[5]; however, unless a strongly prescriptive top-down approach to modelling is being used, there is little specific guidance on the design process.

The aim of the project described in this section was to elicit and acquire some guidelines for the production of an architectural design. The approach used was based on the *probing questions* approach, developed at Rome Air Force Base, USA [Kline & Dolins, 1989] and further developed at AIAI [Inder *et al*, 1990], in which a KBS designer is asked a number of questions about the analysed knowledge, with the answers being used to produce some recommendations of suitable representational techniques. The general format of the questions is:

> IF a certain feature exists in the analysed knowledge
> THEN consider using a particular knowledge representation, or implementation technique.

The designer is asked whether the IF condition is true: if it is, the recommendation supplied by the latter part of the "probing question" is added to the set of recommendations. An example of a probing question might be:

[4]See [Wielinga *et al*, 1993] for a fuller description of the expertise model.

[5]These 3 phases approximately correspond to the stages of *functional decomposition*, *behavioural design* and *physical design* specified in the original KADS project.

IF the problem-solving task is such that a pre-enumerated set of solutions can be established (as distinct from the type of task in which solutions are constructed as a result of the satisfaction of constraints)
THEN use goal-driven reasoning `weighting: 1`
ELSE use data-driven reasoning `weighting: 5`

It can be seen that probing questions are effectively heuristics for knowledge engineers, encoded in a rule-based format. The student's task was to acquire probing questions (either by eliciting knowledge from experienced knowledge engineers, or by reading the available literature), to collate the resulting collection of heuristic rules into a structure of some kind, and then to implement a knowledge based system to run these rules.

3.2 The project: acquisition and analysis

The student chose to concentrate his efforts on eliciting knowledge from experts. Three experts were used, each of whom was interviewed (or asked to make extensive comments on documents sent by electronic mail) on several occasions. Knowledge elicitation techniques used included introductory interviews, card sorting, and triadic comparison of actual KBS systems.

One of the key results of knowledge elicitation was the identification of a number of *functional requirements* and *design features* for knowledge base systems. Many functional requirements relate to the KBS' need to model the domain objects and their relationships, and the need to model the inferences which are made about these domain objects. These requirements may lead to various needs: for example, there may be requirements for certain knowledge representations, for the ability to represent uncertainty in knowledge, and for the ability to generate and examine large numbers of solutions. Design features are knowledge representation and inference techniques used to satisfy the functional requirements. Examples include data-driven reasoning, rules, semantic networks, and blackboard architectures.

There is also a sizeable group of functional requirements, which are concerned with the capabilities of the KBS environment and with producing a smooth interaction between the user and the KBS. These requirements may specify a need for dialogue with and explanations to the user; a need for a high level of computational efficiency in the program; or a need to consider how (or whether) to present the user with large numbers of possible solutions. It was therefore decided that these envirnomental features would also be considered as design features.

The categorisation of functional requirements and design features can be seen in Figure 4. Note that "thoughts of God" is intended to be a catch-all category for ill-defined subjective influences, such as design for elegance, or parsimonious design.

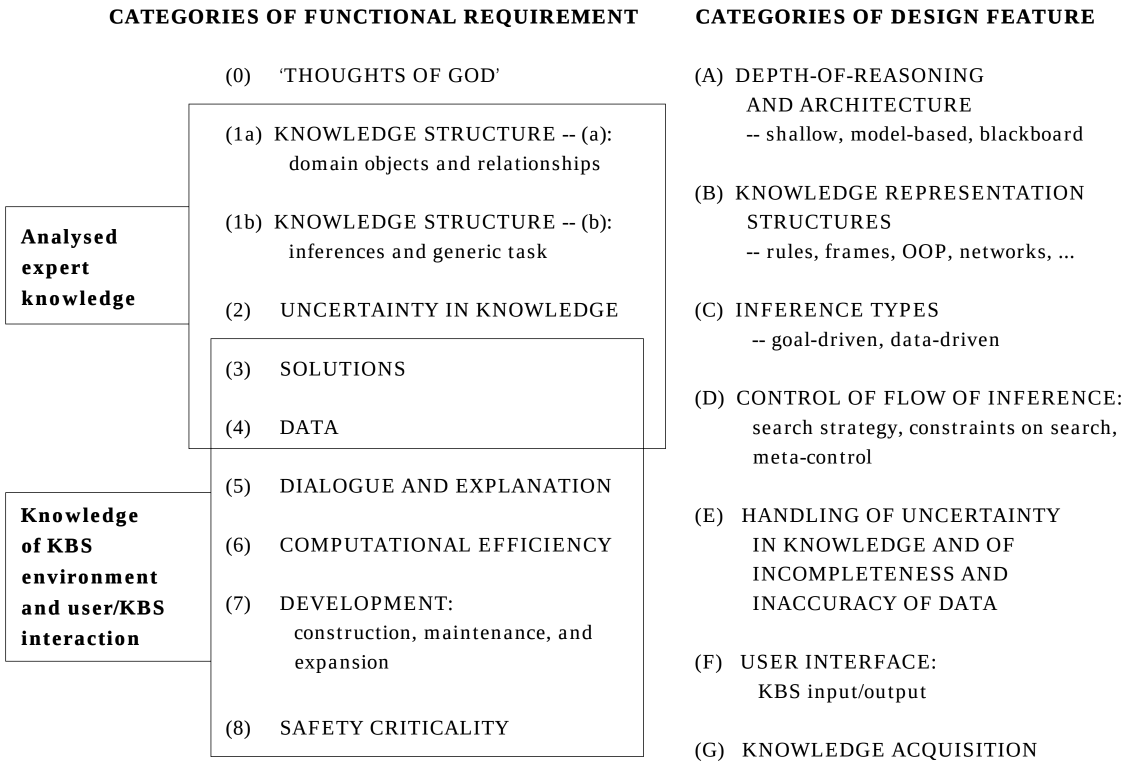
CATEGORIES OF FUNCTIONAL REQUIREMENT
CATEGORIES OF DESIGN FEATURE
(0) 'THOUGHTS OF GOD'
(1a) KNOWLEDGE STRUCTURE -- (a): domain objects and relationships
(1b) KNOWLEDGE STRUCTURE -- (b): inferences and generic task
(2) UNCERTAINTY IN KNOWLEDGE
(3) SOLUTIONS
(4) DATA
(5) DIALOGUE AND EXPLANATION
(6) COMPUTATIONAL EFFICIENCY
(7) DEVELOPMENT: construction, maintenance, and expansion
(8) SAFETY CRITICALITY
Analysed expert knowledge
Knowledge of KBS environment and user/KBS interaction
(A) DEPTH-OF-REASONING AND ARCHITECTURE -- shallow, model-based, blackboard
(B) KNOWLEDGE REPRESENTATION STRUCTURES -- rules, frames, OOP, networks, ...
(C) INFERENCE TYPES -- goal-driven, data-driven
(D) CONTROL OF FLOW OF INFERENCE: search strategy, constraints on search, meta-control
(E) HANDLING OF UNCERTAINTY IN KNOWLEDGE AND OF INCOMPLETENESS AND INACCURACY OF DATA
(F) USER INTERFACE: KBS input/output
(G) KNOWLEDGE ACQUISITION

Figure 4: Categories of functional requirements and design features

The next stage was to create a KADS expertise model to represent all the acquired knowledge from the different viewpoints specified by KADS. In this project, the domain level of the model of expertise was developed first; it was decided that the domain level consisted of the various functional requirements and design features. Once the domain level had been completed, the inference structure was developed. It was decided that the task type being performed was *heuristic classification*. The generic inference structure for heuristic classification and the instantiated inference structure which forms part of the expertise model can be seen in Figure 5. It can be seen that the level of abstraction of the probing questions varies considerably; the conditions might be based on general knowledge of the operation of KBS systems, or on specific knowledge of the task under consideration, and the recommendations might be to use a particular design feature, or to use one of a category of design features.

Finally, a task structure was developed, which specified that the KBS should continue to ask questions until all possible relevant questions had been asked, in order to ensure that all functional requirements are considered.

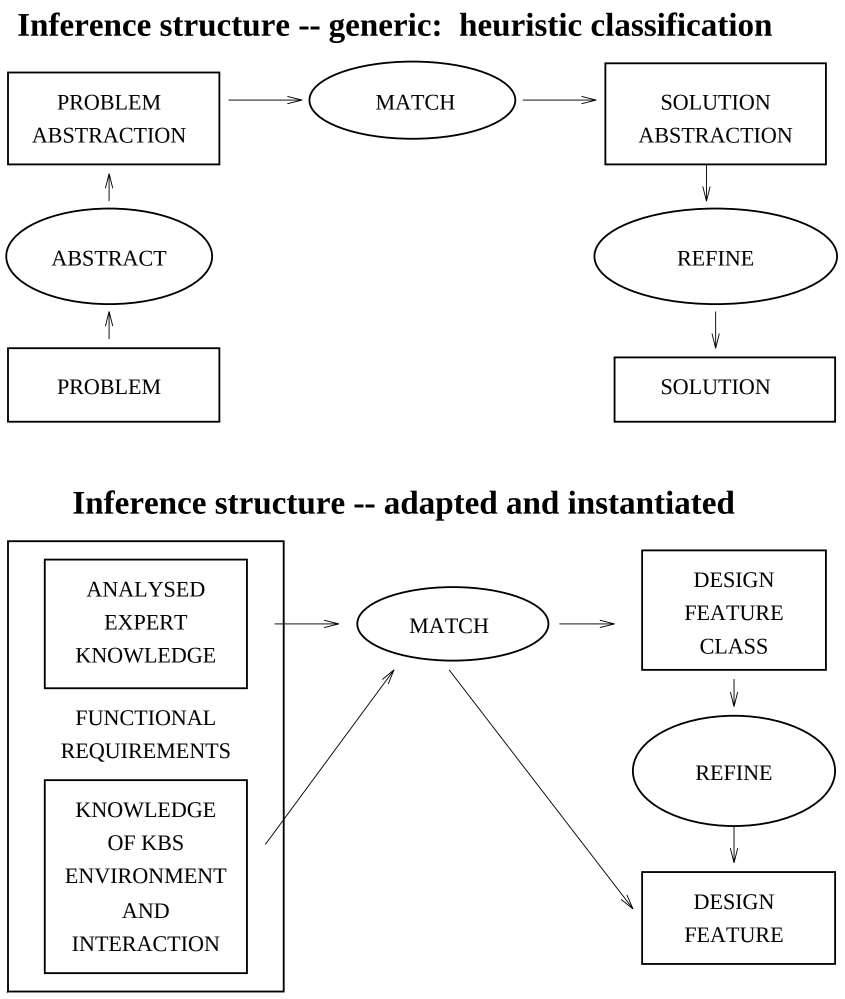
Inference structure -- generic: heuristic classification
PROBLEM ABSTRACTION
MATCH
SOLUTION ABSTRACTION
ABSTRACT
REFINE
PROBLEM
SOLUTION
Inference structure -- adapted and instantiated
ANALYSED EXPERT KNOWLEDGE
FUNCTIONAL REQUIREMENTS
KNOWLEDGE OF KBS ENVIRONMENT AND INTERACTION
MATCH
DESIGN FEATURE CLASS
REFINE
DESIGN FEATURE

Figure 5: Generic and instantiated inference structures for a heuristic classification task

3.3 The project: design

As the design is a relatively late stage in the development of the KBS, the student was able to apply the elicited probing questions to his own design problem. The results of this exercise can be seen in an appendix to the project thesis [MacNee, 1992], in which the final implemented system was (retrospectively) applied to itself. The main resulting recommendations are summarised in the following table:

Design feature	Recommendation
Shallow reasoning	Strong
Rules	Strong
Goal driven reasoning	Moderate
Depth first search	Moderate
Truth maintenance	Moderate
Certainty factors	Moderate
'Canned' text for explanations	Moderate
Data driven reasoning	Weak
Model-based reasoning	Strong negative

It is hardly surprising that a knowledge base which is based around heuristic probing questions should elicit strong recommendations for shallow reasoning and for the use of rules. The justification for some of the other recommendations is less obvious; this was noted during the project, and it was therefore decided that the implemented system would need to be able to justify its reasoning to the user in a clear and coherent manner.[6] An example of an explanation, based on the table above, is that the recommendation for depth-first search arose because of the need to ask questions of the user, and because the student stated that the natural flow of dialogue was to ask detailed questions about one subject before asking general questions about another subject. Depth-first search supports this mode of reasoning, and therefore makes it easy to ask questions in a natural order.

3.4 The project: platform design & implementation

Having performed architectural design, the final stage of design must be performed: the matching up of the recommended design features with the chosen programming tool. The decisions required at this stage are described in more detail in section 4, but for now, it is sufficient to note that the the probing questions should be considered as a starting point for tool selection, not as a prescription. For this project, a choice of two programming tools was available: CLIPS and Prolog. The table above indicates a stronger recommendation for goal-driven reasoning than for data-driven reasoning, which would favour Prolog; however, bearing in mind that the probing questions are heuristics, the factors contributing to this recommendation were examined in more detail. It turned out that the recommendation for goal-driven reasoning was based on a combination of three weak contributing factors; and one of these factors is actually not applicable in this case, because the system is designed to make the KBS ask all possible questions, instead of stopping when a single solution has been found. The recommendation for goal-driven reasoning was therefore downgraded to 'weak'.

[6]This decision accounts for the recommendation for 'canned' text in the above table.

CLIPS and Prolog are both equally strong in most of the other recommended design features (although CLIPS does provide its own facilities for truth maintenance, which Prolog does not); the conclusion is that, from the viewpoint of providing adequate design features, CLIPS and Prolog are both equally recommended for this project. The choice of tool was therefore heavily influenced by other factors, such as the student's previous experience. The tool chosen was CLIPS, a largely rule-based tool whose primary reasoning mechanism is depth-first forward chaining. CLIPS also provides facilities for truth maintenance and certainty factors. Using CLIPS, the PDQ system (which, in this context, stands for "Probing Design Questions") was implemented in approximately 5 weeks.

3.5 Results

PDQ is a workable system which produces recommendations which are helpful, although heuristic, as the above example concerning goal-driven reasoning shows. The questions require the knowledge engineer to have a good understanding of the problem, and some preliminary ideas about possible designs; for example, one question asks if the problem "can be subdivided into 10 or more distinct modules". This suggests that PDQ is most approriate for knowledge engineers who have some experience of building knowledge based systems.

Some further work has been done on the set of probing questions since the PDQ system was completed, in an attempt to separate those questions which produce abstract recommendations from those which recommend more specific design features. This process has lead to the transferral of a small set of probing questions to the first stage of the design process (application design), because some design features (such as blackboard architectures or constraint-based programming) have such a profound effect on the architecture of a program that they must be considered as alternative ways of performing the initial decomposition of the analysed knowledge. The probing questions are therefore able to provide some guidance on whether to perform a structure-preserving design, or whether to make use of an established AI paradigm. This decision is another of the key decisions for knowledge engineers listed in the introduction to this paper. It is possible that the "probing questions" technique could be extended to provide extensive support for this important decision.

4 Choosing a suitable implementation tool

The final project described in this paper aimed to produce guidance on the selection of a suitable shell or toolkit for implementing a knowledge based system [Robertson, 1993].

4.1 The task

The two projects described above have shown that knowledge-based guidance can be provided for certain areas of KADS modelling. However, the knowledge engineer's decision making does not end when the "probing questions" have been answered and the KADS design has been completed; as section 3.4 suggests, the choice of a programming tool requires careful consideration, taking into account both the recommendations of the probing questions and other factors external to the knowledge representation requirements.

The requirements for this project were that the student should identify factors important to the selection of a KBS building tool, and develop a program which would recommend appropriate tools for a project. Given that the PDQ system had already been developed, it seemed sensible to use PDQ's recommendations as a starting point for the identification of important factors. The student also referred to a number of books on knowledge engineering (e.g. [Price, 1990]) which gave their own advice on tool selection.

4.2 The project: acquisition

Knowledge acquisition for this project was initially performed by reading various textbooks, and examining the output of the PDQ system. The results of this work were compiled, and represented graphically using a simple node and arc representation. These diagrams, and associated text, were used as input to a knowledge elicitation interview with an experienced knowledge engineer. This interview produced further useful information, which was used to alter and extend the diagrams. Finally, the student was directed to investigate a set of expert system building tools which was representative of all the categories which he had identified.

The major result of the knowledge acquisition was two classifications of KBS building tools:

- The first classification divided tools into *shells*, *toolkits* and *AI languages*. These categories were further subdivided; shells were divided into those which evaluate rules by pattern matching (e.g. Xi Plus, OPS5, early versions of CLIPS) and those which are effectively "rule networks" (e.g. Crystal).[7] Toolkits were subdivided into top-range toolkits (such as ART and KEE) and mid-range toolkits (such as Nexpert Object, ProKappa and Kappa PC). Languages were not subdivided, since there are comparatively few popular languages for implementing knowledge based systems.

- The second classification was based on the historical background of the tools; tools were classified as belonging to the "ART Camp" (e.g. ART-IM, CLIPS,

[7]This distinction was first defined in [Inder *et al*, 1990].

ECLIPSE) or the "KEE Camp" (including ProKappa and Kappa PC). Tools in the same 'camp' have similar features to each other, since many of them are derived from similar programming philosophies or tools; for example, all tools in the ART camp offer efficient forward chaining rules based on an implementation of the RETE algorithm, but are comparatively weak on backward chaining, because the algorithm used offers almost no support for backward chaining, whereas all tools in the KEE camp have good support for object-oriented programming, but comparatively inefficient rules.[8]

The other main conclusion from the knowledge acquisition phase (largely drawn from [Rothenberg, 1989]) was that the importance of particular features of a programming tool is dependent on the phase of the project. For example, at the very early stages of the project, rapid prototyping might be used to support knowledge acquisition, to investigate the feature of the tool, or to help the knowledge engineer learn to use the tool; at this stage, the training and debugging features of the tool are very important. However, when the final system is delivered, the training and debugging tools are of little or no importance, whereas the speed and efficiency of the execution of the program become very important. The student identified five phases of program development (exploration, prototyping, development, fielding and operation) and seven features of a tool (ease of use, efficiency, extendability, flexibility, portability, reliability and support) which are more or less desirable at various phases. For further details of the proposed relationship between phases of development and tool features, see [Rothenberg, 1989] or [Robertson, 1993].

4.3 The project: analysis

4.3.1 Domain level analysis

The domain level of the expertise model comprised the classifications identified in the knowledge acquisition phase, plus a selection of factors which influence the choice of tool (such as the cost of the tool, the required platform for development, and the experience of programmers with the tool). It also defined a prototype 'frame', with a number of attributes (or, in the terminology of CommonKADS, a **concept**, with a number of **properties**) for defining tools. This frame proved remarkably difficult to define, because of difficulties in distinguishing properties of tools from tool-related concepts. For example, there was considerable discussion on whether the frame should have "forward chaining" as a property (with values such as RETE, PROCEDURAL or NONE) or whether it should have "reasoning types" as a multiple-valued property, with FORWARD CHAINING being one of the values of

[8]See [Mettrey, 1992] for some benchmarking tests. Note, however, that the vendors of KAPPA-PC claim considerable performance improvements in newer versions of KAPPA-PC, which have appeared since Mettrey's study was done.

this property. This discussion led to an important obaservation concerning KADS, which is described in section 4.5.

4.3.2 Inference level analysis

When the inference level is considered, it can be seen that this project is similar in structure to the PDQ project: PDQ identified functional requirements which had to be mapped to design features, whereas this project uses various design features and other requirements to select a suitable tool, or class of tool. However, while the task of PDQ was to identify *all* relevant design features, the task required when selecting a tool is to select the *best* tool for the task. This requires comparison of a set of possible tools against all relevant factors, and assessment of how well each tool matches the overall set of criteria. The most appropriate task type in the KADS taxonomy is therefore *assessment*, rather than heuristic classification.

By the time this project was carried out, some guidance had been published by members of the CommonKADS consortium [Löckenhoff & Valente, 1993] on the configuration of inference structures to match particular assessment tasks. Starting with a minimal generic inference structure (Figure 7), this guidance consists of a set of questions which are asked of the user in order to decide which of the inference steps relevant to assessment tasks are actually performed in this task. The results of these questions are used to devise a problem-specific version of the inference structure (e.g. Figure 8), which is then instantiated to the problem in hand by changing the generic labels on the knowledge roles into problem-specific labels. The final result (Figure 9) forms the inference level of the model of expertise.

A worked example of the configuration of an inference structure can be found in [Kingston, 1993].

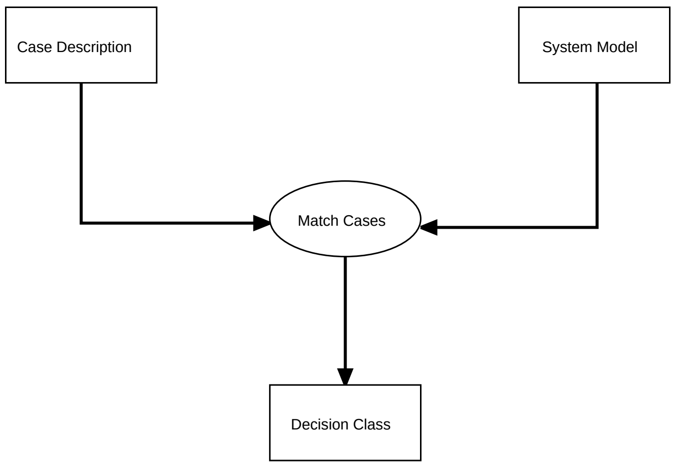

Figure 7: Minimal generic inference structure for assessment tasks

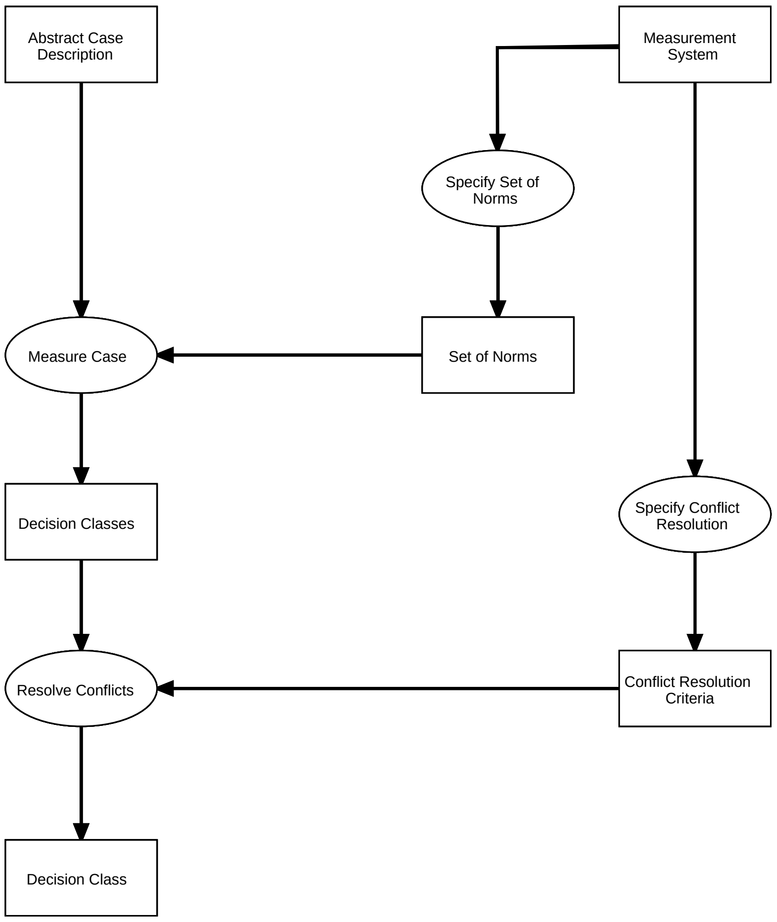

Figure 8: Configured inference structure for selecting a KBS tool

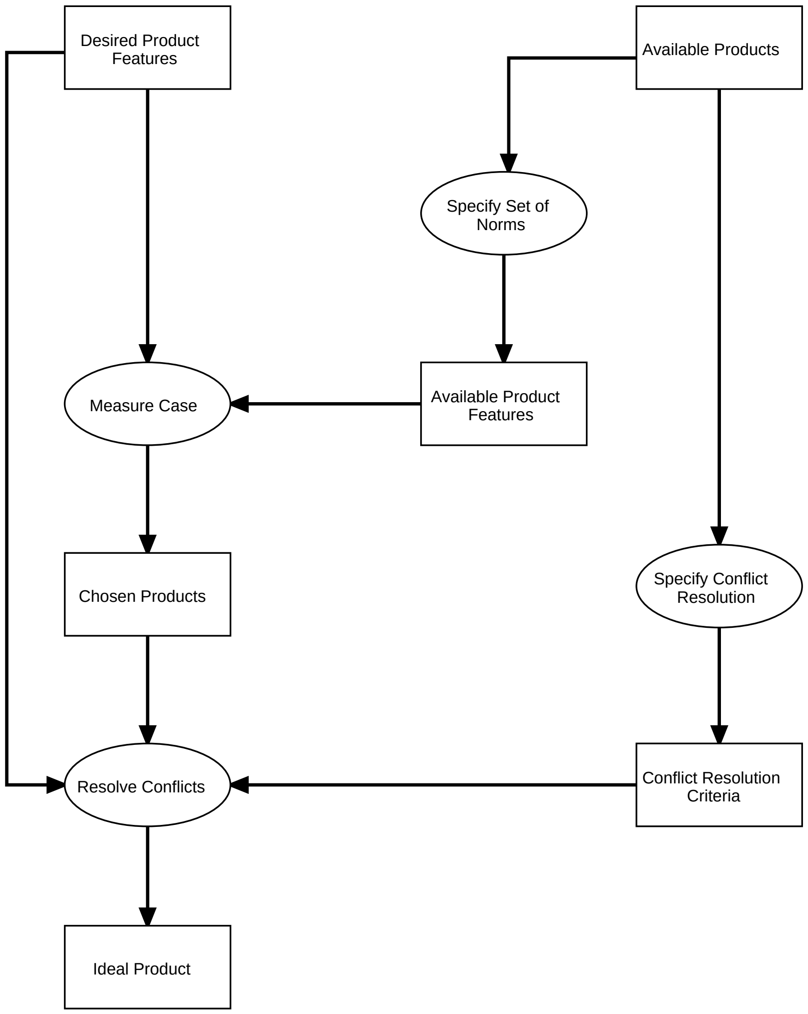

Figure 9: Instantiated inference structure for selecting a KBS tool

It can be seen that the task of selecting a KBS tool requires comparison of the features of available tools against the features desired for a particular project; this produces a shortlist of suggested tools, which is then further refined to choose the ideal tool for the job.

4.4 The project: design and implementation

The configured inference structure contributed greatly to the quick and accurate development of an expertise model. Once the model was complete, design was performed, with the assistance of the PDQ system. PDQ produced very strong recommendations for using both rules (to represent the applicability of different factors) and frames (to represent individual tools); it also produced a strong recommendation for goal-driven reasoning, and a moderate recommendation for data-driven reasoning.

At this point, the student was offered the choice of two tools: CLIPS 5.1 or Sicstus Prolog. The choice was restricted to two tools because these tools were easily available, and because the student had some familiarity with both these tools; the latter factor was important because of the tight timescales of the project. The limited choice simplified the decision process: for reasons described below, Prolog was chosen as the most appropriate tool. The system was therefore implemented using Prolog, using a goal-driven approach which investigated several different categories of tool features one by one. The details of fifteen different KBS tools were also implemented, using a predicate called "frame" which was defined to allow the representation of object-like structures. The features of each tool were then matched against the features required by the user; the relative importance of each feature in the overall assessment was scaled according to the phase of development for which the tool would be used, and according to an importance value entered by the user. The final list of tools, ordered by their overall score, was then displayed to the user.

In order to test the system, it was retrospectively applied to the choice between CLIPS and Prolog for the student's own project. The results of the consultation showed that Sicstus Prolog and CLIPS were among the more highly favoured tools, although they were not at the top of the list; however, most of the tools which were preferred were considerably more expensive. The main factors which led to Prolog being preferred were the recommendation for goal-driven reasoning from the probing questions; the problems in integrating rules and objects in version 5 of CLIPS,[9]; and the student's greater degree of familiarity with Prolog, which led the system to believe that features which were unavailable in Prolog (such as frames) could easily be programmed. It therefore seems that the choice of Prolog was the best decision from the available options.

4.5 Results

This system can be considered to be a sophisticated prototype. Its reasoning capabilities are wide-ranging (over many different tool features and aspects of KBS development) and its algorithm for assessing tools provides results which closely resemble the opinions of expert knowledge engineers; in short, the heart of the system is sufficiently good to be commercially usable. It might find favour as a tool for organisations to assess the adequacy of their existing tools for a proposed project, rather than to recommend a particular too, for a project which is already under

[9]Version 5 of CLIPS included a full object-oriented programming system, but these objects could not be pattern matched in the conditions of rules. Version 6 of CLIPS has introduced this facility.

way. However, its knowledge base needs to be extended to include more tools, and it would benefit from improvements to its user interface and efficiency.

The use of KADS modelling and the probing questions has led to a knowledge base architecture which is well supported by documentation and justification: more importantly it can be extended easily, because new tools can be added to the knowledge base simply by defining a new 'frame'. Ease of maintenance is a vital feature for a system which provides expertise about a domain which changes as rapidly as this one.

5 Discussion

The three projects described above have produced knowledge based systems which provide guidance on particular decisions which users of CommonKADS have to make. The SEXTANT system assists knowledge engineers in the early stages of knowledge analysis, when they are selecting a suitable generic inference structure from CommonKADS' library of inference structures; PDQ provides suggestions for the architectural design of a knowledge based system; and the tool selection system provides guidance on matching the recommended design with available shells or toolkits. The guidance which has been produced seems to be useful: PDQ in particular has been used by later generations of students, and on commercial projects.

It is important to realise that these systems are not intended to take over the decision-making role of a knowledge engineer. The guidance which these systems provide is heuristic; in some cases, careful analysis might suggest that the guidance should not be followed (see section 3.4 for an example). The guidance also requires an understanding of knowledge engineering terminology, as well as an in-depth understanding of the problem to be solved, in order to make the best use of the advice. The key benefit of the guidance is in assisting knowledge engineers by providing a framework for performing the required analysis, just as CommonKADS provides a framework for representing knowledge. It does this by identifying the questions which need to be asked in order to make a particular decision, as well as providing suggested answers to those questions and (in some cases) justification for those answers.

The projects also produced new insights about the modelling techniques recommended by KADS and CommonKADS. The students found that using KADS helped them to think clearly about their knowledge bases and to identify specific areas where problems occurred. However, KADS and CommonKADS are intended to assist rather than to replace the judgement of a knowledge engineer; the students all discovered that the KADS approach did not provide a "magic wand" solution to all the problems which they encountered. For example, the SEXTANT project originally considered the choice of an appropriate task type to be an assessment task, but it was later discovered that a simpler approach, based on a decision tree, could be used to accomplish most of the job. Another example can be seen in the domain modelling for the tool selection project, where the student had difficulty in deciding whether “forward chaining” should be a property or a value of a property. The crux of the problem was that neither approach could be ruled incorrect at a theoretical level; the choice between them depended on purely pragmatic constraints, such as the number of possible values of the property, and the number of possible attributes of the concept. In other words, the definition of concepts and properties in a domain appears to be context-dependent. This observation has far-reaching implications for CommonKADS domain modelling, in that it suggests that there is more than one ‘correct’ model of any chosen domain of knowledge. There is therefore scope for further research on CommonKADS, identifying different styles of domain modelling and the circumstances in which different approaches are most appropriate. In terms of a single project, the choices of classifications at the domain level are unlikely to affect the functionality of the final system very much, although they may affect the ease of achieving that functionality.

6 Summary

In summary, CommonKADS provides a declarative framework for representing knowledge, which can be used to promote clearer thinking and structuring of a knowledge base. CommonKADS is most appropriate for knowledge engineers who have enough experience to be able to make their own decisions about knowledge analysis or design in those exceptional circumstances when CommonKADS' recommendations prove to be unhelpful. CommonKADS itself is sufficiently complex that knowledge engineers require experience with and/or guidance on the KADS approach to use it to its full extent. The projects described in this paper provide guidance for some of the key decisions which must be made when using CommonKADS. This guidance is aimed at those people who would be capable of using CommonKADS: it offers heuristic advice, which is normally good advice but may need to be overridden in exceptional circumstances.

The results of these projects suggest that it is feasible to produce knowledge-based guidance for users of KADS or CommonKADS, as well as producing some new insights about domain modelling and task selection. It is hoped that the results of the projects above can be amalgamated with other projects which are producing guidance of CommonKADS (for tasks such as configuring inference structures [Löckenhoff & Valente, 1993] and converting CommonKADS' Conceptual Modelling Language to its Formal Modelling Language [Aben, 1994]), in order to make CommonKADS a more powerful tool for structuring knowledge engineering.

Acknowledgements

I would like to thank Dr Dave Robertson and Dr Mandy Haggith of the Department of Artificial Intelligence, University of Edinburgh, Ian Filby of AIAI, and Dr Robert Inder of the Human Communications Research Centre, University of Edinburgh for providing the students with supervision and expertise. I would also like to thank Dr Robertson, Dr Haggith and Robert Rae of AIAI for their comments on drafts of this paper.

References

[Aben, 1994]	Aben, M. (1994). *Formal methods in Knowledge Engineering.* Unpublished Ph.D. thesis, SWI, University of Amsterdam, The relevant chapter is also available as CommonKADS report KADS-II/T1.2/WP/UvA/040/1.0.
[Breuker., 1987]	Breuker., J. A. (1987). *Model-driven Knowledge Acquisition.* University of Amsterdam and STL, ESPRIT project 1098, Deliverable A1.
[deHoog *et al*, 1993]	de Hoog, R., Martil, R., Wielinga, B., Taylor, R., Bright, C. and van de Velde, W. (1993). The Common KADS model set. ESPRIT Project P5248 KADS-II KADS-II/M1/DM1.1b/UvA/018/ 5.0, University of Amsterdam and others.
[Inder *et al*, 1990]	Inder, R., , Aylett, R., Bental, D., Lydiard, T. and Rae, R. (October 1990). Study on the Evaluation of Expert Systems Tools for Ground Segment Infrastructure: Final Report. Technical Report AIAI-TR-84, AIAI.
[Kingston, 1993]	Kingston, J.K.C. (1993). Re-engineering IMPRESS and X-MATE using CommonKADS. In *Expert Systems 93.* British Computer Society, Cambridge University Press. Also available as AIAI-TR-130.
[Kingston, 1994]	Kingston, J.K.C. (1994). Design by Exploration: A Proposed CommonKADS Inference Structure. *Submitted to 'Knowledge Acquisition'.*

[Kline & Dolins, 1989]	Kline, P.J. and Dolins, S.B. (1989). *Designing expert systems : a guide to selecting implementation techniques*. Wiley.
[Krueger, 1992]	Krueger, A. (Sept 1992). Classification of Expert Tasks: the SEXTANT system. Unpublished M.Sc. thesis, Dept of Artificial Intelligence, University of Edinburgh.
[Löckenhoff & Valente, 1993]	Löckenhoff, C. and Valente, A. (March 1993). A Library of Assessment Modelling Components. In *Proceedings of 3rd European KADS User Group Meeting*, pages 289–303. Siemens, Munich.
[MacNee, 1992]	MacNee, C. (Sept 1992). PDQ: A knowledge-based system to help knowledge-based system designers to select knowledge representation and inference techniques. Unpublished M.Sc. thesis, Dept of Artificial Intelligence, University of Edinburgh.
[Mettrey, 1992]	Mettrey, W. (February 1992). Expert systems and tools: Myths and realities. *IEEE Expert*.
[Price, 1990]	Price, C. (1990). *Knowledge Engineering Toolkits*. Ellis Horwood, This book concentrates on available toolkits, and on how to go about selecting an appropriate toolkit. While some of the tools it mentions are now somewhat dated, the principles of tool selection are still valid. It also gives clear examples of implementation.
[Robertson, 1993]	Robertson, S. (Sept 1993). A KBS to advise on selection of KBS tools. Unpublished M.Sc. thesis, Dept of Artificial Intelligence, University of Edinburgh.
[Rothenberg, 1989]	Rothenberg, J. (1989). Expert System Tool Evaluation. In Guida, G. and Tasso, C., (eds.), *Topics in Expert System Design*. North-Holland.
[Schreiber *et al*, 1993]	Schreiber, A. Th., Wielinga, B. J. and Breuker, J. A., (eds.). (1993). *KADS: A Principled Approach to Knowledge-Based System Development*. Academic Press, London.

[Tansley & Hayball, 1993] Tansley, D.S.W. and Hayball, C.C. (1993). *Knowledge-Based Systems Analysis and Design: A KADS Developers Handbook.* Prentice Hall.

[van de Velde *et al*, 1993] van de Velde, W., Duursma, C. and Schreiber, G. (1993). Design model and process. Unpublished KADS-II/M7/MM/VUB, Vrije Universiteit Brussel.

[Wielinga, 1993] Wielinga, B. (October 1993). Expertise Model: Model Definition Document. CommonKADS Project Report, University of Amsterdam, KADS-II/M2/UvA/026/2.0.

[Wielinga *et al*, 1992] Wielinga, B., Van de Velde, W., Schreiber, G. and Akkermans, H. (1992). The KADS Knowledge Modelling Approach. In *Proceedings of the Japanese Knowledge Acquisition Workshop (JKAW'92).*

[Wielinga *et al*, 1993] Wielinga, B., van de Velde, W., Schreiber, G. and Akkermans, H. (Jun 1993). Expertise Model Definition Document. CommonKADS Project Report, University of Amsterdam.