Selecting a KBS tool using a knowledge based system

Stefan Robertson, Midland Treasury Support Services, London, England
John K.C. Kingston, AIAI, University of Edinburgh, Scotland

AIAI-TR-214

February 1997

This paper is an extended version of a paper which was presented at the Joint Pacific Asian Conference on Expert Systems / Singapore International Conference on Intelligent Systems, Orchard Hotel, Singapore, 24-27 February 1997.

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

Abstract

When an organisation embarks onto a project or strategy which will bring knowledge based systems into their business processes, one of the most frequently asked questions is, "Which software tool should be used"? The task of selecting the best tool requires both an understanding of the relevant characteristics of a project and a considerable knowledge of the competing alternatives, from which it was deduced that this selection task could usefully be implemented using a knowledge based system.

The purpose of this paper is to describe the nature of the task of tool selection, and to describe how the relevant aspects of this task were represented using the CommonKADS methodology in order to encode them in a knowledge based system. The finished system was tested by applying it to the task of selecting the best tool for implementing itself. The paper concludes with a discussion of the important factors in selecting a KBS tool, and the appropriateness of CommonKADS for this project.

1 Introduction

When an organisation embarks onto a project or strategy which will bring knowledge based systems into their business processes, one of the most frequently asked questions is, "Which software tool should be used"? There are many sofware toolkits available for programming knowledge based systems, offering a wide variety of programming techniques, and selling for an equally wide range of prices. The task of selecting the best tool requires both an understanding of the relevant characteristics of a project and a considerable knowledge of the competing alternatives, from which it was deduced that this selection task could usefully be implemented using a knowledge based system. This task was duly undertaken in the summer of 1993, using knowledge elicited from experienced KBS programmers on the staff of the University of Edinburgh.

The purpose of this paper is to describe the nature of the task of tool selection, and to describe how the relevant aspects of this task were represented in order to encode them in a knowledge based system. This project was carried out using part of the CommonKADS methodology for developing knowledge based systems [Tansley & Hayball, 1993] [Schreiber *et al*, 1993] [Kingston, 1992], which helped to identify certain important aspects of the task; the key models from the CommonKADS analysis are therefore also described. It is hoped that this paper will serve as a guiding document for anyone who is trying to decide which tool to purchase, as well as describing the implementation of a functioning knowledge based system. A fuller description of the project can be found in [Robertson, 1993].

2 KBS Tools

The first stage in the project was to acquire knowledge about different tools, and the factors which affect the selection of KBS tools. This was done by reading product brochures, by performing knowledge elicitation sessions with a member of AIAI staff, and by reading product reviews (such as [Mettrey, 1992]) and literature (such as [Price, 1990]). A *KBS tool* is taken to be a software package designed to assist in the development of expert systems. These differ from knowledge elicitation tools and "knowledge engineering workbenches", which are designed to assist in knowledge elicitation and modelling. KBS tools provide an environment within which expert systems can be prototyped, developed and implemented.

There are approximately 200 commercially available KBS tools, and so the first step in understanding the products available was to attempt to determine if they could be grouped together in some fashion. Initial knowledge acquisition suggested that the most useful dimensions for classification are price, functionality and platform, and that the first two are closely related (based upon the observation that additional functionality is usually tied to additional cost). Most authors use a broadly similar classification; for example, [Alty, 1989] groups the products into three main categories based primarily on functionality, which also happen to differ markedly in the platforms on which they are available:

- Shells
- Languages
- Toolkits

[Price, 1990] has a similar classification. However he breaks down toolkits into three subgroups; *loosely coupled*, *high-end* and *medium-priced* toolkits. Loosely coupled toolkits are simply products with a selection of tools complete with an underlying language. High-end and medium-priced tools are classified based upon the relationship between price and functionality.

After performing a knowledge elicitation session, the classification described below was decided upon, based on the representational features offered by each tool.

2.1 Shells, toolkits and languages

2.1.1 Shells

Shells are the smallest and simplest of all KBS tools. They tend to run on PCs, often requiring kilobytes rather than megabytes of RAM. They are a lot cheaper than the larger tools, with a top price of around £1000. They are also easier to learn and use (knowledge representation is largely restricted to the use of rules), and so can be used by people without many AI or programming skills.

On the negative side, shells are more restrictive in the types of tasks that they can undertake. Memory constraints and the lack of certain knowledge representations (such as objects) means that more complex knowledge-based problems such as planning or design problems may not be able to be handled with these products.

Shells can be broken into two further categories; *Rule Based Network Shells* and *Pattern Matching Shells*. Rule based network shells are the most restrictive type; their rules cannot contain any variables, with the result being the rules effectively form a decision tree. The usual inference method in these shells is backward-chaining. A good example of a rule network shell is CRYSTAL, from Intelligent Environments.

Pattern matching shells are usually less restrictive. Rules can contain variables, with forward or backward chaining provided (although usually not both). The main feature of this classification is the pattern-matching network present in the tools. This network is designed to speed up the inference process. A good example of a pattern matching shell is OPS5.

2.1.2 Toolkits

Toolkits are much more powerful than shells. Representational power is increased by providing both rules and objects. Toolkits tend to require powerful machines, allowing the development of more complex systems than shells can handle easily. Some toolkits offer both forward-chaining and backward-chaining of rules. However, the increased power means that the level of expertise of the developer needs to be higher, in both programming and AI terms; this is often because experience is needed in order to select the best features for the task at hand.

Toolkits, like shells, can be broken into two main subcategories: *Mid-range Toolkits* and *Top-range Toolkits*. Mid-range toolkits tend to require 386 PCs or more powerful machines; the price usually falls in the range between £1000 and £5000. They are typically written in C, making them reasonably efficient. These toolkits allow programming using both rules and objects. Top-range toolkits require powerful workstations; originally this meant dedicated LISP machines, but now UNIX workstations can be used. Most top-range toolkits are still written in Lisp. A key feature of top-range toolkits is the provision of some form of assumption-based truth maintenance system [de Kleer, 1986]; this allows the creation of multiple *what-if* scenarios. Top-range toolkits also allow the use of forward and backward chaining rules as well as objects, and a number of other features.

2.1.3 AI Languages

Languages are the most flexible tools of all. They can cater for any type of task that can be represented on a computer. However, what is gained in representational power is also lost in the increased time required to develop systems.

Unlike the previous two categories, shells and toolkits, languages do not provide any sort of inference engine[1]. Nor is any knowledge representation formalism (rules or objects) provided, and so one must be developed[2]. These factors contribute to a longer development time and a need for expensive expertise.

2.2 ART-like and KEE-like

A second dimension for classifying tools was also identified. This groups the products into various camps based on whether the product is similar to ART or KEE. Inference ART©and KEE©were among the first commercially successful toolkits. For some time, they were almost the only commercially available tools to offer both rules and objects integrated withing a single package. Many of the tools that followed tended to copy the features of either ART or KEE, thus producing the "ART Camp" and the "KEE Camp".

2.2.1 ART Camp

Tools in this category are based around the rule as the prime representational mechanism. All products in this camp have a efficient forward-chaining inference method based upon the Rete network. This network dramatically improves the speed of the inference engine. Backward-chaining is provided by a few tools, implemented on top of the forward-chaining engine.

Most ART-like tools provide objects, but these tend to be secondary to the rule system. A limited form of graphics is sometimes supplied.

Examples of tools in the ART Camp are:

- Inference ART
- Clips
- Eclipse
- OPS5
- ART/Enterprise[3]

[1]The exception is Prolog, which provides a simple goal driven, depth-first inference engine.
[2]Again, the Common Lisp Object System (CLOS) provides an exception to the rule)
[3]formerly called ART-IM

2.2.2 KEE Camp

Tools in the KEE camp use objects as their main representation method. Rules are usually provided, but they tend to be less efficient than the ART camp. Both forward and backward-chaining inference, usually on the same rules, are provided.

A powerful feature of many tools in the KEE camp is *active images*; graphics which are linked to objects. When the object is altered in any way, this alteration is reflected in the display; conversely, if the graphic image is altered by the user, the underlying object is also updated.

Examples of tools in the KEE Camp are:

- KAPPA-PC
- KEE
- Object-IQ
- ProKappa

2.3 Other important features of KBS tools

It is clear from the discussion in the previous section that the representational features offered by a tool – i.e., support for rules and/or objects, forward and/or backward chaining – are the most important features to consider when choosing a KBS tool. However, the knowledge elicitation on this project identified a number of other factors which should be considered. For example, facilities for supporting uncertain or hypothetical reasoning may be useful; several shells support numerically based uncertain reasoning (often by attaching Bayesian probabilities to assertions) while the top-range toolkits provide support for full-scale assumption-based truth maintenance systems [de Kleer, 1986]. The interfaces which a tool supports are also very important; this category includes facilities for constructing user interfaces, the user interface of the tool itself, and interfaces to external programs (e.g. databases or spreadsheets). There are also a large number of other features to be considered, such as

- price (both development and delivery versions);
- required hardware and software;
- vendor support;
- quality of documentation associated with the tool;
- any features which the tool might provide for securing knowledge bases.

For a fuller list of features, see [Stylianou *et al*, 1995].

For this project, the large number of possible features which could be considered were summarised using categories derived from [Rothenberg, 1989]. These categories were:

- Flexibility;
- Ease of Use;
- Efficiency;
- Extendability;
- Support.

In addition, two other categories were explicitly identified. These were:

- Portability;
- Reliability.

2.3.1 Flexibility

This refers to the range of knowledge representation techniques in a KBS tool. The larger the number of techniques, the more flexible the product. The implication of this is that top-range toolkits are usually more flexible than mid-range toolkits, mid-range toolkits more flexible than pattern matching shells, and pattern matching shells more flexible than rule network shells. AI languages are the most flexible tools of all.

2.3.2 Ease of Use

This attempts to measure how usable the product is. This factor is inversely related to the number of features provided by a tool, because the more features a tool has, the harder it is to use. However, the quality of the interfaces provided by a tool can greatly affect the ease of use.

It is important to note that the user of a tool will vary over the course of the development life-cycle. During exploration, prototyping, and development, the knowledge engineer is the user; during fielding and operation, the end-user becomes the system user. The quality of the interfaces provided for the knowledge engineer and the interfaces which can be produced by the knowledge engineer are therefore both germane to ease of use.

Tool	Time (sec.)	Ratio
Art (2.0)	1.2	1
Kee (2.1)	165	138

Table 1: Relative Speeds of Art and Kee

2.3.3 Efficiency

The efficiency of the tool relates mainly to the speed of execution. It was found that the most important efficiency issue, particularly for tools which use forward-chaining rules, is the presence of a pattern matching network. The aim of this network is to reduce the number of matching operations that need to be performed during each recognise-act cycle. It does this by building a network of all the possible matches for the rule base. This network is only altered when facts are asserted or retracted.

The most common form of pattern matching network is the Rete algorithm. A full description can be located in [Forgy, 1992]. This is the algorithm present in the Art Camp products, and represents a substantial speed differential over those products which do not have such a network.

This claim can be demonstrated with reference to a test carried out by [Riley, 1986]. This saw the monkey-and-bananas problem carried out on a Symbolics Lisp machine with ART, with a Rete network, and KEE, without a Rete network. The results can be seen in Table 1; ART outperformed KEE by more than two orders of magnitude. (N.B. Both ART and KEE have progressed by several versions since this test was performed, so the absolute timings should not be considered valid for current versions of these tools).

There are a number of other issues affecting the efficiency of a product. Products written in C tend to have a slight speed advantage over those written in languages such as Lisp. In addition, the presence of a knowledge base compiler can improve the speed of inference by generating a representation in the underlying language.

2.3.4 Extendability

This is the degree to which any system developed using the KBS tool can be extended. The extension can encompass a variety of avenues. These include increasing the size of the knowledge base, interfacing with external programs and receiving data from outside sources (databases and spreadsheets).

2.3.5 Support

The primary issue in this category is the level of support provided by the vendor. This covers areas like *telephone hot-lines*, *training services* and *consultancy*. In addition the stability of the vendor and their commitment to the product must be factored in.

The history of Expertech Ltd. provides a good example of the importance of vendor stability and commitment to a product. Expertech produced the pattern matching shell Xi Plus, which sold fairly well in the 1980s. Expertech then took over the operations of another vendor, ISI, who sold a shell named SAVOIR and a mid-range toolkit named Egeria. Expertech ceased to support SAVOIR (since it was a competitor to Xi Plus) but continued to market Egeria; vendor stability was therefore a major factor in the demise of SAVOIR. Some years later, Expertech merged with the Inference Corporation to become Inference Europe Ltd. Inference Europe no longer market Egeria (which competes with their mid-range toolkit, ART/Enterprise), and their clear commitment to ART-related products raises questions about their commitment to marketing and supporting Xi Plus.

2.3.6 Portability

The important issues here concern the runtime options of the product and the delivery hardware that will be utilised. If the product is only designed to run on a single machine, then this sharply reduces the portability of that product. A separate version of the product would have to be purchased for each production machine. The highest degree of portability is exhibited by those products which supply an compiler to produce an executable version of the system.

2.3.7 Reliability

A crude measure of reliability might be the history of the product. If a product has had a large amount of previous development, then it could be considered more reliable than a product which has only had a little use. This tends to favour those products that have been around longer, because the developers have had more time and (presumably) more demand to iron out most of the problems.

There is not much of a need for a reliability measure for most commercial KBS tools; these products are well-tested, commercially successful products, and are therefore believed to be reliable. If an organisation considers some of the shells available in the public domain, then reliability might become a far greater issue.

3 Tool Selection

So how is tool selection performed? A KBS tool needs to be suitable for:

1. the nature of the problem;
2. the phase of development;

3. the policy and capabilities of the organisation;
4. the budget for the project;
5. the development team;
6. the available hardware.

If more than one tool proves suitable, then a good heuristic is to used the simplest tool possible:

> "*Use a shell when you can,*
> *a toolkit when you should,*
> *and a language if you must.*" [Barrett & Beerel, 1988]

3.1 Suitability for the Nature of the Problem

It is well known amongst AI researchers that some knowledge-based tasks are easier than others to encode in knowledge-based systems, and that more difficult problems generally require more powerful tools. As a result, various attempts have been made to classify knowledge-based tasks (e.g. [Price, 1990] [Stylianou *et al*, 1995]), and to understand what makes a task easy or difficult for a knowledge-based system. One of the most useful classifications is that provided by the CommonKADS methodology for KBS development, which categorises tasks as *system analysis* tasks, *system synthesis* tasks or *system modification* tasks. The full hierarchy can be seen in Figure 1. As the category names imply, system analysis tasks involve the analysis of an existing system or state, system synthesis tasks require the creation of a new system or state, and system modification requires both analysis (of a non-optimal state) and synthesis (of an improved state). In general, system analysis tasks are the simplest tasks for knowledge based systems to tackle, and system modification tasks are the hardest.

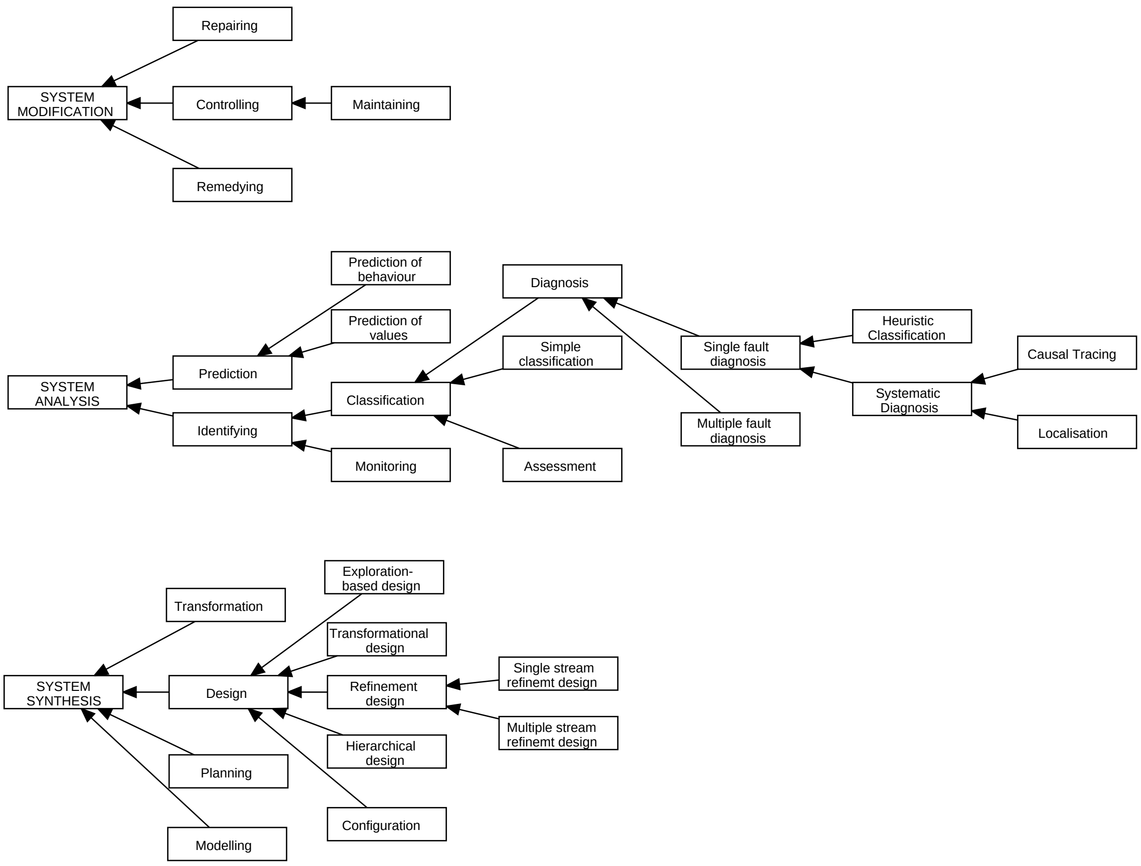

Figure 1: The CommonKADS taxonomy of task types

However, the task type is not the only feature of a problem which affects the suitability of KBS tools. This can be illustrated by examining three well-known expert systems in the medical domain: MYCIN, INTERNIST and CASNET (see [Johnson & Keravnou, 1985]). These three systems all perform the same task (medical diagnosis), but require significantly different levels of functionality in their programs. MYCIN is entirely rule-based, INTERNIST makes heavy use of objects, and CASNET uses FORTRAN to implement a causal model of symptoms and diseases. In fact, the functionality which is desirable for a particular problem depends on a large number of features of the problem; so many that an entire book has been filled with "probing questions" [Kline & Dolins, 1989], which are statements which identify one or more features of a problem, and suggest suitable functionality based on that feature. An example of a probing question is given below.

> On average, do we know five or more new facts about a domain object simply by being told that it is of type X?

OR

Are these new facts not known with certainty, but assumed unless there is evidence to the contrary?

Yes → Place the object in a data structure (e.g. *frames, semantic nets* or *objects*) whose *inheritance mechanism* will provide the facts when needed, and whose *default values* will be assumed unless an exception is specifically asserted.

No → Assert the new facts explicitly, which is a 'cheap' solution.

AIAI has its own set of probing questions, based on the expertise of its own staff as well as on published work (see [MacNee, 1992] or [Kingston, 1995]). The use of these questions to identify necessary functionality for a project has been integrated with the final stages of CommonKADS modelling. It was assumed that the project for which a tool was being selected had already been subjected to probing questions analysis, and that the recommendations of the probing questions could be taken as input to this system.

3.2 Suitability for the Phase of Development

A number of factors which affect the choice of product depend upon the phase of development within which the product will be used. Rothenberg [Rothenberg, 1989] suggests the following five phases of development in the lifecycle of a typical KBS:

- Exploration;
- Prototyping;
- Development;
- Fielding;
- Operation.

3.2.1 Exploration

This phase is also known as conceptualisation. The focus is upon the support provided by the tool for the task of defining a problem. This definition can involve the clarification of the problem scope, identifying the main issues and even some broad design.

The most important factors in this phase are flexibility and ease of use. Flexibility is important, as the developer may need to utilise many different knowledge representation techniques.

3.2.2 Prototyping

This is phase where one or more implementations of the system are developed rapidly. The purpose of these implementations is to refine the problem successively and to test the merits of different approaches.

Like the previous phase, flexibility is an important factor. The very nature of prototyping demands a wide range of techniques, whose features can be adopted and tested. Support from the vendor and product for the developer tend to be important. The developer may be trying relatively new techniques. This may require training and debugging tools.

3.2.3 Development

The main purpose of this phase is to provide a functional system. The requirements for the system can emerge from a prototyping exercise, in-depth analysis, or a combination of the two. At the completion of this phase, the system should be ready for delivery to user.

At this point in the development life cycle of a product, design decisions become locked in. This reduces the importance of flexibility. Reliability, ease of use and support tend to be the more dominant factors.

3.2.4 Fielding

Until this phase, the main user of the product has been the system developer. The end-user may have been given access to some of the early prototypes with the aim of providing guidance for future development.

This phase sees the end-user playing a more important role. During this phase, the system is moved from a development environment into the end-user, or delivery, environment. This makes the ease of use factor concerned with user interfaces, not the development environment. This movement from development to production increases the importance of portability.

3.2.5 Operation

The day-to-day running of the delivered system is the main function of this phase. The end-user is the main operator of the system. The only time the developer would be active in this phase, is for the purpose of maintenance.

Reliability, efficiency and ease of use are the most important issues in this phase. Unless the users are satisfied with all these factors, then the development may not be viewed as successful.

3.2.6 Relationship between Development Phases and Tool Features

In order to bring the phases and factors together, the relationship between them can be summarised in Figure 2. This graph is similar to one in [Rothenberg, 1989]. It differs in the addition of the two factors; portability and reliability.

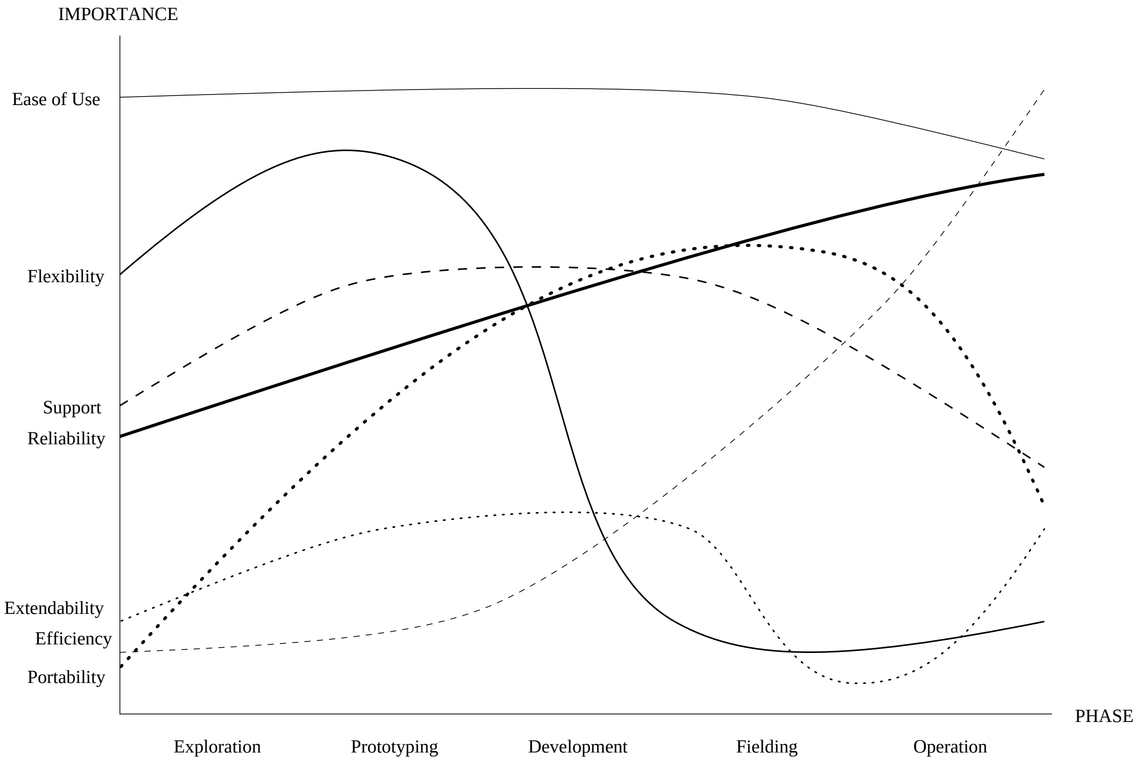

Figure 2: The Relationship between Factors and Development Phases

3.3 Suitability for the Organisation

Organisations do have policies which can restrict the choice of product, vendor or hardware. The restriction on products and vendors may be due to some special price arrangement, preferring products which have a substantial development history, or strategic policy. Organisations also have staff with certain skills – and, typically, budgets with certain limitations. These must be considered when selecting a KBS tool; indeed, they may prove to be the overriding factor in selection in many cases.

3.4 Cost

Although the cost of a product is dependent upon development phase, it was not included in the previous section. As a project progresses, overall costs increase. The main commitment to cost occurs at the transition between phases. At this point, management must commit itself to the requirements of the following phase. It is this always increasing, transitional nature of cost that causes it to be handled separately.

The price of the product is not the only impact upon cost. The need to purchase additional hardware, training and consultancy charges all have a substantial impact on the overall cost of a project. This data is much harder to obtain. For the purposes of simplicity, this project only considered the product price when evaluating cost.

3.5 Development Team

The makeup of the development team can have a big impact on the choice of product. There are two main areas to consider when evaluating the team. These are expertise and preferences.

3.5.1 Team Expertise

Expertise can take three forms. The level of AI experience can have an impact on the choice of product. The top-range toolkits do require a substantial amount of AI experience. This is because of the wide variety of features offered. Experience is needed to select the best features for the task at hand.

Some products do not deliver all the features that are required. However, it may be possible to program those features. For example, none of the top-range toolkits support the use of certainty factors within rules, but this feature can be programmed in if desired. The team's ability to program these features must be considered.

The third, and final, form is previous experience with the products. If any member of the team has had experience with a certain product, then this may be a case for choosing it. This is because less training and familiarisation will be required.

3.5.2 Team Preference

Some notice should be taken of the preferences of the development team. The team may prefer to use a certain product over others. This is not as important as expertise, but it should be considered.

3.6 Hardware

Products are supplied to run on a fixed set of hardware platforms. Some products do have differing development and delivery platforms, but most offer a common platform.

Shells and mid-range toolkits tend to run on PCs. The top-range toolkits are available on Unix Workstations or Lisp machines. A few products are now available for the IBM Mainframe.

4 Analysis of the problem using CommonKADS

The categorisation of the tools and the identification of the factors which influence tool selection completed the knowledge acquisition for this project. The next stage was to analyse the knowledge acquired using the various models prescribed by the CommonKADS methodology. The heart of this process is the creation of an *expertise model*, which represents knowledge about a problem at three levels:

- *Domain* knowledge is declarative knowledge about objects, states and relationships;
- *Inference* knowledge is procedural knowledge about the processes which are required to solve the problem;
- *Task* knowledge describes the control over these processes.

4.1 Building the Expertise Model

4.1.1 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 'object', with a number of attributes (or, in the terminology of CommonKADS, a **concept**, with a number of **properties**) for defining tools. This object 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 this property. Eventually, it was decided that more detailed information could be encoded if "forward chaining", "backward chaining" etc. were encoded as individual properties.

4.1.2 Inference level analysis

When the inference level is considered, it can be seen that this project uses various design features and other requirements to select a suitable tool, or class of tool. 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 CommonKADS taxonomy (shown in Figure 1) is therefore *assessment*.

The next stage in CommonKADS is usually to obtain the generic inference structure for assessment tasks from the CommonKADS library of inference structures. This generic inference structure would then be modified and instantiated to represent the processes which are actually carried out on the project. However, some guidance has been published by members of the CommonKADS consortium [Löckenhoff & Valente, 1993] [Valente & Löckenhoff, 1994] on altering the generic inference structure for assessment tasks of inference structures, so that it resembles the activities carried out in a particular project more closely. This guidance consists of a series of questions which the knowledge engineer asks himself about the application; a positive answer to a question causes an inference step to be replaced by one or more inference steps and knowledge roles. A worked example of this configuration process can be found in [Kingston, 1993].

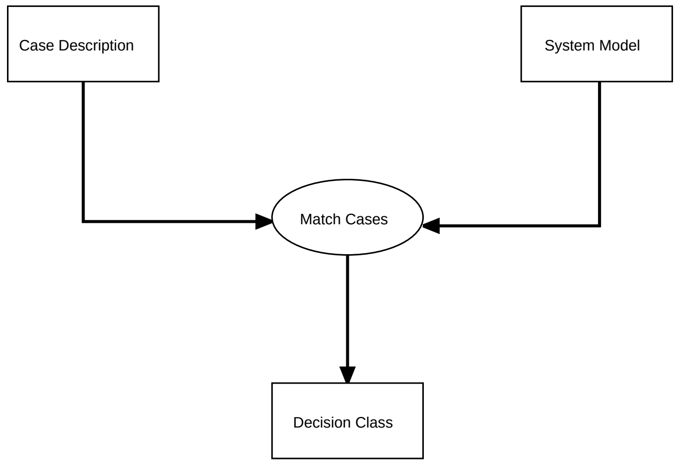

Figure 7: Generic inference structure for assessment tasks

For this project, the starting point was the generic inference structure for assessment tasks shown in Figure 7. The configuring questions were then applied. The first two questions determine whether the case description needs to be *abstracted* before it is matched, and whether the system model needs to be *specified* before it is matched.

- Is the case description already abstract enough to be matched?
  Yes.
  The description of the desired expert system building tool will be based upon the system model. This will ensure that the case is in the correct format.

- Is the system model already specific to be matched?
  Yes.
  The measurement system will be explicit in the system model. The input knowledge role will be compared directly against the system model, identifying those products that could fulfill the task.

The fact that the answers to both the above questions was "yes", means that no further inference steps (ellipses) are needed to represent abstraction of the case description or specification of the system model.

The next problem is to determine the "inner contents" of the **match** inference step. CommonKADS suggests three different approaches to matching. These are *matching (only)*, *matching plus computation* and *matching plus conflict resolution*. The choice is again guided by configuration questions:

- Is the decision class the direct result of matching the case against the measurement system?
  No.
  The matching process could yield many products which match the features of the input case. Some further pruning is required.

- Is the decision class the result of computation?
  No.
  Some form of conflict resolution will be required to distinguish between the many potential products.

The answers to these questions suggest that the third approach (matching plus conflict resolution) is the approach being used to solve this problem. In order to represent this in the inference structure, the **match** inference step is replaced by two **specify** inference steps, one **measure** inference step and one **resolve** inference step, with intermediate knowledge roles. In addition, the **system model** knowledge role is renamed to **measurement system**, for the sake of clarity. The end result is the inference structure shown in Figure 8.

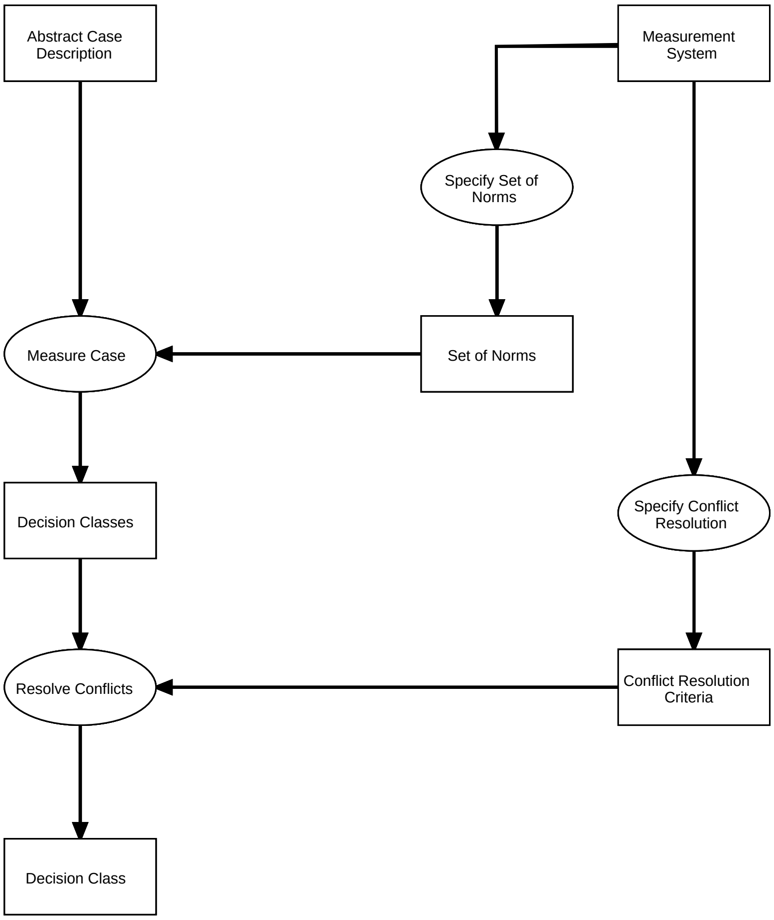

Figure 8: Configured inference structure for selecting a KBS tool

This inference structure must then be instantiated to the domain; in other words, the knowledge roles are given domain-specific names, indicating which domain knowledge they represent. Any domain-specific knowledge which is not yet represented as a knowledge role, or domain-specific links between knowledge roles and inference steps, are also identified at this stage and added to the inference structure. This produces the inference structure which becomes part of the Expertise Model for this project. This structure can be seen in Figure 9.

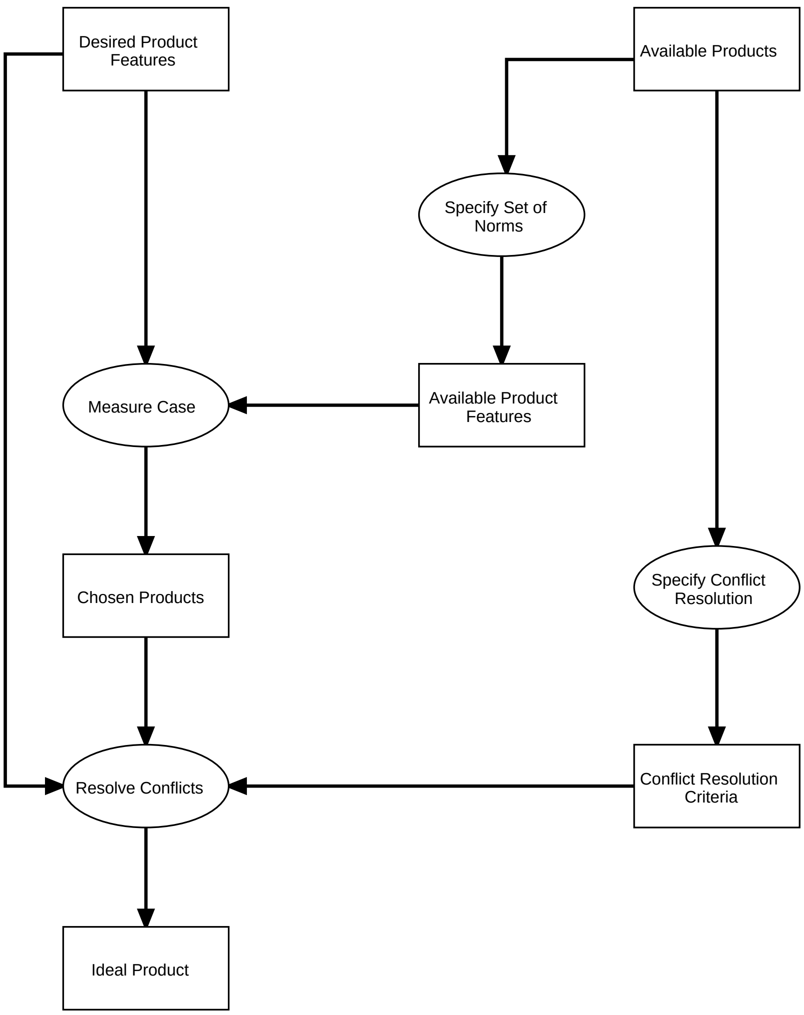

Figure 9: Instantiated inference structure for selecting a KBS tool

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.

4.1.3 Task level analysis

The third level of a CommonKADS Expertise Model comprises the *task* knowledge; that is, control knowledge about partial ordering of inference steps, input and output of individual inferences, and alternative methods for achieving the same goal. It was decided that this level would not be developed in great detail for this project; the inference structure was considered to sufficiently simple for detailed specification of control knowledge to be unnecessary.

4.2 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 AIAI's set of probing questions. These questions suggested a number of techniques:

- Frames/objects (to represent individual tools);
- Inheritance (between tools in the same category e.g "shells", "ART camp tools");
- Rules (to represent the applicability of different factors);
- Canned text (for explanation);
- Data-driven reasoning;
- Goal-driven reasoning.

The recommendations for both data-driven reasoning and goal-driven reasoning are possibly contradictory, and are therefore worthy of further investigation. It turns out that data-driven reasoning was proposed as the result of the following answers to probing questions:

- The number of solutions exceeds the number of data required;
- Solutions are not mutually exclusive;
- The user is seeking all possible solutions.

Goal-driven reasoning was recommended because of the following answers to other probing questions:

- Solutions can be pre-enumerated;
- Tasks can be decomposed into sub-tasks;
- Data is gathered by questions and answers;
- Reasoning will be presented to the user on request.

It seemed that either data-driven or goal-driven reasoning could be utilised in the solution to this problem; both have advantages and, perhaps more importantly, none of the probing questions which indicate major disadvantages for certain reasoning types (e.g. goal driven reasoning for a synthetic task) were triggered. It may even be the case that both could be used, in differing parts of the system.

At this point, it was necessary to choose a software tool for this project The postgraduate student who carried out this project was offered a choice of two tools (CLIPS 5.1 and Sicstus Prolog), because these two tools (plus Lisp) were the only tools easily available to students. 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 features of fifteen different KBS tools were entered into the system, using a Prolog predicate called "frame" which allowed the representation of object-like structures. The attributes of these 'objects' reflected the features of tools, as identified in the domain knowledge of the Expertise model. The features of each tool were then matched against the features required for a KBS project; the relative importance of each feature in the overall assessment was scaled according to the phase(s) 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. As further information was obtained, the importance scores were updated.

In order to test the system, it was retrospectively applied to the choice of the most suitable tool for this project, considering all 15 tools in the system's knowledge base[4]. The results of the consultation showed that Sicstus Prolog and CLIPS were among the more highly favoured of the tools considered, although they were not at the top of the list; however, the tools which were more highly recommended were considerably more expensive. Prolog was preferred to CLIPS, because:

- There are problems in integrating rules and objects in version 5 of CLIPS,[5];
- The student's greater familiarity with Prolog suggested that features which were unavailable in Prolog (such as object-like structures) could easily be programmed;
- The PDQ system [MacNee, 1992] [Kingston, 1995], which automates probing questions analysis, suggested that goal-driven reasoning was slightly preferred to data-driven reasoning.

It therefore seems that the choice of Prolog was the best decision from the available options.

[4]These tools were chosen as a representative sample of the many tools which are available.

[5]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.

5 Discussion

It can be seen that the problem of selecting a suitable KBS tool is amenable to representation as a knowledge-based assessment task, and to implementation within a knowledge-based system. The task does appear to be an assessment task (one where the best solution is chosen from a number of possibilities) rather than a classification task (e.g. categorising every tool as “suitable” or “unsuitable”). It's also important to note that the representational features offered by the tool are important, but are by no means the only factors to be taken into account; for example, this project showed that the policies of an organisation can greatly restrict the range of choices, and that familiarity with a tool can often compensate for a lack of certain representational features.

The system could usefully be extended in a number of ways:

- The categorisations of products could be used to develop a concept of 'similar' products. This could be used to determine how easy it would be for a knowledge engineer to learn a new product; for example, a knowledge engineer who has experience with CLIPS would find it easier to learn to use ART/Enterprise than to learn how to use KEE.

- A package could be developed to support maintenance of the knowledge base; a user interface could provide easy access to the 'frames', and a chronological sorting function could notify the user of frames which have not been updated for some time.

- Further features could be added to the list of features considered. For example, the ability for knowledge-based tools to be embedded into other products, or at least to communicate via DLLs or DDE, is becoming increasingly important.

The use of CommonKADS 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'. However, the difficulties experienced in deciding on appropriate properties for a tool (*reasoning types* vs *forward chaining*) highlight an important factor in CommonKADS domain modelling: the distinction between concepts and properties is dependent on the viewpoint taken. In other words, an item of knowledge (such as *forward chaining*) which is classified as a property from one viewpoint may be classified as a value of a property (i.e. a concept) from another viewpoint. This implies that reusability of domain knowledge in CommonKADS may prove difficult, since a new knowledge-based task must share not only a domain, but also a viewpoint, with an existing knowledge-based task.

References

[Alty, 1989] Alty, J. L. (1989). Expert system building tools. *Topics in Expert System Design.*

[Barrett & Beerel, 1988] Barrett, M. L. and Beerel, A. C. (1988). *Expert Systems in Business: A Practical Approach.* Ellis Horwood.

[de Kleer, 1986] de Kleer, J. (1986). An assumption based truth maintenance system. *Artificial Intelligence*, 28.

[Forgy, 1992] Forgy, C. (September 1992). Rete: A fast algorithm for the many-pattern/many-object pattern-match problem. *Artificial Intelligence*, 19(1).

[Johnson & Keravnou, 1985] Johnson, L. and Keravnou, E. (1985). *Expert Systems Technology: A Guide.* Abacus Press, Cambridge, Mass. 02139.

[Kingston, 1992] Kingston, J. (1992). Pragmatic KADS: A methodological approach to a small knowledge based systems project. *Expert Systems: The International Journal of Knowledge Engineering.*

[Kingston, 1993] Kingston, J.K.C. (1993). Re-engineering IMPRESS and X-MATE using CommonKADS. In *Research and Development in Expert Systems X*, pages 17–42. Cambridge University Press. http://www.aiai.ed.ac.uk/ jkk/publications.html.

[Kingston, 1995] Kingston, J. K. C. (February 1995). Applying KADS to KADS: knowledge based guidance for knowledge engineering. *Expert Systems*, 12(1).

[Kline & Dolins, 1989] Kline, P. J. and Dolins, S. B. (1989). *Designing expert systems : a guide to selecting implementation techniques.* Wiley.

[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. (September 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. J. (1990). *Knowledge Engineering Toolkits*. Ellis Horwood.

[Riley, 1986] Riley, G. D. (1986). Timing tests of expert-system-building tools. US government memo, Lyndon B. Johnson Space Center, NASA, Houston.

[Robertson, 1993] Robertson, S. (September 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. *Topics in Expert System Design*.

[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.

[Stylianou *et al*, 1995] Stylianou, A. C., Smith, R. D. and Madey, G. R. (1995). An Empirical Model for the Evaluation and Selection of Expert System Shells. *Expert Systems with Applications*, 8(1):143–156.

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

[Valente & Löckenhoff, 1994] Valente, A. and Löckenhoff, C. (1994). Assessment. In Breuker, J. and van de Velde, W., (eds.), *The CommonKADS Library*, chapter 8. IOS Press.

A A Sample Session

The following is a script of a session with the Tool Selection program. The purpose of this session was to choose a platform with which to develop the program itself.

```
Tool Selection Program
======================

The purpose of this program is to assist in the selection
of an Expert System shell, toolkit or language for use in
the development and/or delivery of an expert system.

The following products are evaluated:
          level_5
          personal_consultant_easy
          experience
          crystal
          clips
          eclipse
          leonardo_level_1
          sicstus_prolog
          austin_kyoto_common_lisp
          personal_consultant_plus
          nexpert_object
          prokappa
          art
          kee
          muse


Enter "help" for a list of all the available commands

Do you wish to restrict the evaluation to a subset
of these products?

|: no
Knowledge Representation
========================
This refers to the way "expert knowledge" is
encoded by the product. Usually a product may
have more than one method available. It is left
to the developer to decide which method to choose.

Please enter the features you require within this class. (One per line)
Each feature MAY be followed by a measure of importance.
This importance must be an integer between 1 and 5.
          -> 5 is "very important" or required
          -> 1 is "not important" or desired
If no importance is entered, it defaults to 3.

What knowledge representation features do you require?

|: options

The following options are available:

    rule_based_representation
    frames
    objects
    meta_knowledge
    knowledge_sources
```

```
    facts

What knowledge representation features do you require?

|: rule_based_representation 4

What knowledge representation features do you require?

|: objects 4

What knowledge representation features do you require?

In the area of knowledge_representation, you requested:
          rule_based_representation importance 4
          objects importance 4

Inference Type
==============
This is the direction of inference employed by
the product. It can proceed from the data to the
goal, from the goal back to the data, or a combination
of the two.

Please enter the features you require within this class. (One per line)
Each feature MAY be followed by a measure of importance.
This importance must be an integer between 1 and 5.
          -> 5 is "very important" or required
          -> 1 is "not important" or desired
If no importance is entered, it defaults to 3.

What inference types do you require?

|: goal_driven_reasoning 3

What inference types do you require?

|: data_driven_reasoning 2

What inference types do you require?

In the area of inference_type, you requested:
          goal_driven_reasoning importance 3
          data_driven_reasoning importance 2



Inference Control
=================
This is the search strategy and control mechanism
used by the Product. They affect the order states
are examined during the inference process.

Please enter the features you require within this class. (One per line)
Each feature MAY be followed by a measure of importance.
This importance must be an integer between 1 and 5.
          -> 5 is "very important" or required
          -> 1 is "not important" or desired
If no importance is entered, it defaults to 3.

What inference control techniques do you require?

In the area of inference_control, you requested no features.
```

```
Uncertainty
===========
The methods employed to represent uncertainty
in the knowledge.

Please enter the features you require within this class. (One per line)
Each feature MAY be followed by a measure of importance.
This importance must be an integer between 1 and 5.
          -> 5 is "very important" or required
          -> 1 is "not important" or desired
If no importance is entered, it defaults to 3.

What types of uncertainty do you require?

In the area of uncertainty, you requested no features.

System Interface
================
The mechanisms available to the product for
interaction with external systems.

Please enter the features you require within this class. (One per line)
Each feature MAY be followed by a measure of importance.
This importance must be an integer between 1 and 5.
          -> 5 is "very important" or required
          -> 1 is "not important" or desired
If no importance is entered, it defaults to 3.

What system interface features do you require?

In the area of system_interface, you requested no features.



Product                    Score                    Change
=======                    =====                    ======
nexpert_object             13                       13
prokappa                   13                       13
art                        13                       13
kee                        13                       13
experience                 11                       11
clips                      10                       10
leonardo_level_1           10                       10
level_5                     9                        9
eclipse                     9                        9
sicstus_prolog              9                        9
personal_consultant_plus    9                        9
muse                        9                        9
personal_consultant_easy    7                        7
crystal                     7                        7
austin_kyoto_common_lisp    4                        4


Limitation Phase
================

The quality of the delivered representational mechanisms
must be taken into account.
NOTE: No scaling of this phase takes place.

Do you require variables in the rules?

|: y
```

```
What is the estimated size of the rulebase?

|: 50


Product                  Score                   Change
=======                  =====                   ======
prokappa                 14                       1
kee                      14                       1
nexpert_object           13                       0
art                      13                       0
experience                9                       -2
eclipse                   9                       0
sicstus_prolog            9                       0
personal_consultant_plus  9                       0
muse                      9                       0
level_5                   7                       -2
clips                     7                       -3
personal_consultant_easy  5                       -2
crystal                   5                       -2
leonardo_level_1          5                       -5
austin_kyoto_common_lisp  4                       0



|: why change in clips

Product: clips
Phase: limitation

     Description                                  Influence
     ===========                                  =========

     REQUESTED: rule_based_representation
     REQUESTED: objects
     object_limitation(pattern_matching)             -2

     REQUESTED: objects
     object_limitation(multiple_inheritance)         -1

Task Phase
==========

The details of the task to be performed with this product
are considered in this phase. The answers given in this
phase, tend to affect a product, depending on which class
of expert system building tool it is.

How important do you consider the task?

|: 3


What size range of rulebase are you anticipating?

|: small


Can all the solutions to the problem be pre-enumerated?

|: y
```

```
Does the task involve large amounts of simultaneous reasoning?

|: n


Do you require a special representation language?

|: n


Does the task require a real time interface?

|: n


Product                   Score                  Change
=======                   =====                  ======
prokappa                  14                     0
kee                       14                     0
nexpert_object            13                     0
art                       13                     0
experience                11.4                   2.4
eclipse                   11.4                   2.4
level_5                    9.4           2.4
clips                      9.4           2.4
sicstus_prolog             9                     0
personal_consultant_plus   9                     0
muse                       9                     0
personal_consultant_easy   7.4           2.4
crystal                    7.4           2.4
leonardo_level_1           7.4                   2.4
austin_kyoto_common_lisp   4                     0


Cost Phase
==========

This phase considers the approximate amount of money
you are prepared to spend on a product.

How important do you consider cost?

|: 1


What is the maximum price you would like to pay for a tool?

|: 5


Do you want to exclude more expensive products?

|: n

*** no change in scores ***

Vendor Phase
============

A company may have Vendors they prefer to deal with.
```

```
How important do you consider the vendor?

|: 1


Please enter the names of any preferred vendors?

*** no change in scores ***
Hardware Phase
==============

A project may be restricted in the choice of hardware it
has available to use. This phase attempts to take this
into consideration for both development and delivery.

How important do you consider hardware?

|: 4


What development phases will this product be
required to perform?

|: exploration

What development phases will this product be
required to perform?

|: prototyping

What development phases will this product be
required to perform?

|: fielding

What development phases will this product be
required to perform?


What platform will be used for development?

|: unix_workstation

What platform will be used for development?


Do you want to exclude all products without the
required hardware?

|: n


What delivery platform will be utilised?

|: unix_workstation

What delivery platform will be utilised?


Product                  Score                  Change
=======                  =====                  ======
prokappa                 14                      0
kee                      14                      0
```

```
art                        13                       0
nexpert_object             13                       0
eclipse                    11.4                     0.0
sicstus_prolog              9                       0
muse                        9                       0
experience                  8.2                     -3.2
clips                       7.8                     -1.6
level_5                     6.2                     -3.2
personal_consultant_plus    5.8                     -3.2
personal_consultant_easy    4.2                     -3.2
crystal                     4.2                     -3.2
leonardo_level_1            4.2                     -3.2
austin_kyoto_common_lisp    4                       0
```

```
Team Phase
==========

The experience and preferences of the development staff
are an important consideration. If a product does not
deliver requested features, but they can be programmed -
then the importance for that feature is incorporated.

How important do you consider the team?

|: 4


What products would the development team prefer to use?

|: sicstus_prolog

What products would the development team prefer to use?

|: clips

What products would the development team prefer to use?


What products do the development team have experience with?

|: sicstus_rΦ Φprolog

What products do the development team have experience with?

|: clips

What products do the development team have experience with?

|: austin_kyoto_common_lisp

What products do the development team have experience with?


What level of AI experience does the development
team possess?

|: 5
```

```
What programming skills do the team possess?


Product                   Score                   Change
=======                   =====                   ======
prokappa                  14                       0
kee                       14                       0
art                       13                       0
nexpert_object            13                       0
sicstus_prolog            12.2                     3.2
eclipse                   11.4                     0.0
clips                     11.0                     3.2
muse                       9                       0
experience                 8.2                     0.0
level_5                    6.2                     0.0
personal_consultant_plus   5.8                     0.0
austin_kyoto_common_lisp   5.6                     1.6
personal_consultant_easy   4.2                     0.0
crystal                    4.2                     0.0
leonardo_level_1           4.2                     0.0


Ease of Use Phase
=================

This is the "user friendliness" of the product. The user
of the product changes as the development life cycle
progresses. The developer is the main user in the early
stages, with the end-user being important later.

The following phase is scaled according to its importance
in the development phases in which the product will be
operating.

How important do you consider usability?

|: 1


Do you require a graphic interface for the users?

|: n


Product                   Score                   Change
=======                   =====                   ======
prokappa                  14.8                    0.8
kee                       14.8                    0.8
art                       13.8                    0.8
nexpert_object            13.8                    0.8
sicstus_prolog            12.2                    0.0
eclipse                   11.4                    0.0
clips                     11.0                    0.0
muse                       9.44                   0.44
experience                 9.2                    1.0
personal_consultant_plus   6.6                    0.8
level_5                    6.4                    0.2
austin_kyoto_common_lisp   5.6                    0.0
personal_consultant_easy   5.2                    1.0
crystal                    5.2                    1.0
leonardo_level_1           4.4                    0.2
```

```
Efficiency Phase
================

This phase attempts to take into account the efficiency
of the product. Important issues are:
     KB Compilers
     Type of forward chaining mechanism
     Whether the program is written in C


The following phase is scaled according to its importance
in the development phases in which the product will be
operating.

How important do you consider efficiency?

|: 3


Product                    Score                Change
=======                    =====                ======
prokappa                   15.57               0.77
kee                        15.18               0.38
art                        14.18               0.38
nexpert_object             14.18               0.38
sicstus_prolog             12.58               0.38
eclipse                    12.55               1.15
clips                      11.77               0.77
muse                       10.17               0.77
experience                  9.58               0.38
level_5                     7.17               0.77
personal_consultant_plus    6.98               0.38
austin_kyoto_common_lisp    5.6                0.0
personal_consultant_easy    5.58               0.38
crystal                     5.58               0.38
leonardo_level_1            5.17               0.77



|: why change in sicstus_prolog

Product: sicstus_prolog
Phase: efficiency

     Description                                 Influence
     ===========                                 =========

     The product was written in C                1

     Development phase - exploration.
          SCALE: 0.1

     Development phase - prototyping.
          SCALE: 0.2

     Development phase - fielding.
          SCALE: 0.5

     Program phase importance - average.
          SCALE: 0.6
                                                 =========
     CHANGE FACTOR                               0.38
```

```
|: why change in clips

Product: clips
Phase: efficiency

     Description                                      Influence
     ===========                                      =========

     REQUESTED: rule_based_representation
     rete_forward_chaining                            2

     The product was written in C                     1

     REQUESTED: goal_driven_reasoning
     rete_forward_chaining
     Art Camp                                         -1

     Development phase - exploration.
          SCALE: 0.1

     Development phase - prototyping.
          SCALE: 0.2

     Development phase - fielding.
          SCALE: 0.5

     Program phase importance - average.
          SCALE: 0.6
                                                      =========
     CHANGE FACTOR                                    0.77
Extendability Phase
===================

This phase considers the ability of any system developed
with product to be extended. The important issues are:
     external program interfaces
     rulebase size limitations
     database interfaces
     spreadsheet interfaces
     integration with other systems


The following phase is scaled according to its importance
in the development phases in which the product will be
operating.

How important do you consider extendability?

|: 1


Product                    Score                 Change
=======                    =====                 ======
prokappa                   16.30               0.73
kee                        15.18               0.0
nexpert_object             14.92               0.74
art                        14.43               0.25
sicstus_prolog             13.08               0.49
eclipse                    12.55               0.0
clips                      12.38               0.62
muse                       10.41               0.25
experience                 10.2                0.62
personal_consultant_plus    7.72               0.74
```

```
level_5                  7.66               0.49
personal_consultant_easy 6.2                0.62
crystal                  5.83               0.25
leonardo_level_1         5.78               0.62
austin_kyoto_common_lisp 5.6                0.0


Flexibility Phase
=================

The number of knowledge representation methods available
in a product contribute to its flexibility. The larger
the number, the more flexible.

The following phase is scaled according to its importance
in the development phases in which the product will be
operating.

How important do you consider flexibility?

|: 4


Product                 Score               Change
=======                 =====               ======
kee                     18.99              3.81
art                     18.24              3.81
prokappa                17.83              1.52
nexpert_object          17.21              2.29
sicstus_prolog          16.12              3.05
clips                   14.67              2.29
eclipse                 13.31              0.76
muse                    12.7               2.29
experience              11.72              1.52
personal_consultant_plus 10.01             2.29
austin_kyoto_common_lisp  8.65             3.05
level_5                   8.42             0.76
leonardo_level_1          7.31             1.52
personal_consultant_easy  6.96             0.76
crystal                   6.59             0.76


Portability Phase
=================

This phase considers the ability of a system developed
on this product to ported from one machine to another.

The following phase is scaled according to its importance
in the development phases in which the product will be
operating.

How important do you consider portability?

|: 1


Do you intend to implement any systems developed with
this product on multiple machines?

|: n
```

```
*** no change in scores ***

Reliability Phase
=================

An attempt to measure how reliable a product is.
NOTE: There is currently no information in the KB
regarding reliability.

The following phase is scaled according to its importance
in the development phases in which the product will be
operating.

How important do you consider reliability?

|: 1


Product                   Score                  Change
=======                   =====                  ======
kee                       19.18                 0.19
art                       18.43                 0.19
prokappa                  17.83                 1.52
nexpert_object            17.21                 2.29
sicstus_prolog            16.12                 3.05
clips                     14.67                 2.29
eclipse                   13.31                 0.76
muse                      12.7                  2.29
experience                11.72                 1.52
personal_consultant_plus  10.01                 2.29
austin_kyoto_common_lisp   8.65                 3.05
level_5                    8.42                 0.76
leonardo_level_1           7.31                 1.52
personal_consultant_easy   6.96                 0.76
crystal                    6.59                 0.76




Support Phase
=============

Support is broken down into two main parts:
     (a) vendor assistance
     (b) developer assistance from product
The need for support diminishes as the development life
cycle progresses.

The following phase is scaled according to its importance
in the development phases in which the product will be
operating.

How important do you consider support?

|: 1


Product                   Score                  Change
=======                   =====                  ======
kee                       20.52                 1.34
art                       19.77                 1.34
prokappa                  18.98                 1.15
nexpert_object            18.16                 0.96
```

```
sicstus_prolog            16.31                   0.19
clips                     14.7                    0.0
eclipse                   13.89                   0.57
muse                      13.65                   0.96
experience                12.49                   0.76
personal_consultant_plus  11.15                   1.15
level_5                    9.38                   0.96
austin_kyoto_common_lisp   8.66                   0.0
personal_consultant_easy   7.92                   0.96
leonardo_level_1           7.70                   0.38
crystal                    6.97                   0.38


Selecting the Best Product
==========================

Product                   Score
=======                   =====
kee                       20.52
art                       19.77
prokappa                  18.98
nexpert_object            18.16
sicstus_prolog            16.31
clips                     14.7
eclipse                   13.89
muse                      13.65
experience                12.49
personal_consultant_plus  11.15
level_5                    9.38
austin_kyoto_common_lisp   8.66
personal_consultant_easy   7.92
leonardo_level_1           7.7
crystal                    6.97


The best product for the situation is - kee

Product:     kee
Version:     3.1
Vendor:      intellicorp
Cost:        10000 usd

KEE, short for Knowledge Engineering Environment, is
one of the most powerful toolkits on the market.
It has a wide range of representation techniques:
including rules, frames, objects meta knowledge and
a truth maintenance system. It provides support for
both forward and backward-chaining. Its rules
are less efficient than those of ART, as no Rete
network is employed. A fully developed developer
and user interface is provided. The object system
is linked to the user interface by means of an active
graphic facility, called ActiveImages.



To see how a product arrived at its score;
              enter "how <product name>"

|: how sicstus_prolog
sicstus_prolog had the following requested features:

Feature class - knowledge_representation
```

```
    Feature                            Importance
    -------                            ----------
    rule_based_representation          4

Feature class - inference_type

    Feature                            Importance
    -------                            ----------
    goal_driven_reasoning              3
    data_driven_reasoning              2



Product: sicstus_prolog
Phase: team

    Description                                     Influence
    ===========                                     =========

    team_preference(sicstus_prolog)                2

    product_experience(sicstus_prolog)             2

    Program phase importance - above_average.
         SCALE: 0.8
                                                    =========
    CHANGE FACTOR                                   3.2


Product: sicstus_prolog
Phase: efficiency

    Description                                     Influence
    ===========                                     =========

    The product was written in C                    1

    Development phase - exploration.
         SCALE: 0.1

    Development phase - prototyping.
         SCALE: 0.2

    Development phase - fielding.
         SCALE: 0.5

    Program phase importance - average.
         SCALE: 0.6
                                                    =========
    CHANGE FACTOR                                   0.38


Product: sicstus_prolog
Phase: extendability

    Description                                     Influence
    ===========                                     =========

    external_program_interface                      2

    integrated                                      2
```

```
     Development phase - exploration.
          SCALE: 0.2

     Development phase - prototyping.
          SCALE: 0.4

     Development phase - fielding.
          SCALE: 0.2

     Program phase importance - low.
          SCALE: 0.2
                                                  =========
     CHANGE FACTOR                                0.49


Product: sicstus_prolog
Phase: flexibility

     Description                                  Influence
     ===========                                  =========

     rule_based_representation                    1

     Languages are very flexible                  3

     Development phase - exploration.
          SCALE: 0.7

     Development phase - prototyping.
          SCALE: 0.8

     Development phase - fielding.
          SCALE: 0.2

     Program phase importance - above_average.
          SCALE: 0.8
                                                  =========
     CHANGE FACTOR                                3.04

Product: sicstus_prolog
Phase: support

     Description                                  Influence
     ===========                                  =========

     REQUESTED: rule_based_representation
     rule_tracer                                  1

     Development phase - exploration.
          SCALE: 0.5

     Development phase - prototyping.
          SCALE: 0.7

     Development phase - fielding.
          SCALE: 0.7

     Program phase importance - low.
          SCALE: 0.2
                                                  =========
     CHANGE FACTOR                                0.19
```