The Development, Validation and Implementation of Knowledge-Based Systems

Robert M. O'Keefe

Decision Sciences and Engineering Systems

School of Management, Rensselaer Polytechnic Institute

Troy, NY 12020, USA.

Computer Laboratory

University of Kent at Canterbury

Kent CT2 7NF, England, U.K.

Alun D. Preece

Department of Computing Science

University of Aberdeen, King's College

Aberdeen AB9 2UE, Scotland, U.K.

Abstract

It is now 10 years since Operational Research, and other groups that support management decision making, seized upon Knowledge-based Systems (KBS) as an alternative set of techniques and methods for building systems that support, augment or automate decision making. Success, as with most information technologies, has been mixed. This paper briefly reviews where KBS have been successful, and what benefits have been derived. It notes that interest in KBS has been rekindled by the present interest in business process re-engineering.

Given the mature state of development tools, most KBS developers are now focusing on methodological and life-cycle issues. The paper reviews progress in understanding and using development methodologies, validation methods (now generally considered as crucial to technical success) and implementation. Some ideas about where KBS "fits in" to the array of tools, techniques and methodologies available to the modern analyst and system developer are presented.

Keywords: knowledge-based systems, expert systems, decision support, system development, validation, implementation.

1. Introduction

It is now ten years since the first author (O'Keefe, 1985) and others (Holroyd et al., 1985, Kastner and Hong, 1984) introduced expert systems to the world of Operational Research (OR). As an alternative way of doing modelling, building models with qualitative and heuristic components as opposed to mathematical components, expert systems offered a fresh avenue worth exploring. In a subsequent paper (O'Keefe, Belton and Ball, 1986), reporting the experiences of a research project that attempted to use expert systems from an OR perspective, one of us wrote:

> "We hope that ultimately the development of expert systems will become an established part of OR, perhaps as fundamental as linear programming, simulation, statistics ... There is enough evidence to suggest that expert systems provide a useful method of developing decision support systems for management."

Like so many quotes seen with hindsight, this appears naive. Expert systems applications have met with both success and failure. Impediments to success have been technological (including difficult problems in acquiring, representing and validating knowledge) and methodological (including difficulties in project management, promoting user acceptance, and avoiding unrealistic user expectations). Since the mid-eighties, interest in expert systems has visibly waned, and then somewhat recovered due to new technical approaches (e.g., case-based reasoning and constraint-based reasoning) and association with some "hot" application areas and management buzz-phrases (especially business process re-engineering and knowledge-based software engineering).

At the end of this paper, we will (perhaps foolishly) reword and restate this quote, hoping that it does not have to be revised again ten years hence! Prior to this, we briefly review present expert systems practice. The review is unashamedly methodologically focused — it

assumes that technical tools are delivered by others, normally in the form of software, and that the analyst is allowed to concentrate on development, validation and implementation of the system. It is also very personal, reflecting our own experience and research. Before we can discuss methodology, some review of terminology, the state of technology, and the key application areas over the past ten years is presented.

2. Terminology

Defining an expert system is still as problematic as it was ten years ago. The preferred term has become knowledge-based system (KBS), in part because so much of the knowledge that goes into many KBS is not derived from an expert, but also because the phrase "expert system" came to be over-associated with so-called "first generation" rule-based systems.

So, what is a KBS? To be called a KBS, we think three minimal conditions must be met. First, there must be some explicit, identifiable knowledge base (KB); the developer should be able to point to some text or objects and claim it to be the knowledge (note that, for this to be possible, the KB must not be inter-mixed with control and reasoning procedures). Second, the knowledge and reasoning processes should be in some part qualitative, that is, dealing with semantic objects that require definition. Third, the KBS should perform at least some of its inference based upon dynamic changes in knowledge; the execution order of the system should not be defined solely by data, as with many algorithms.

This very loose definition does not say anything about technology, since it has become apparent that KBS tools can be used to implement many types of programs, and a piece of software is *not* a KBS just because it is developed using a KBS tool. For example, one can construct a program using a rule-based programming language where (a) the knowledge and control of execution is inter-mixed, (b) all the rules perform quantitative transformations, and (c) the execution order of the rules is pre-determined. Such a program meets none of the

above minimal requirements, yet we have seen a number of pieces of rule-based software that perform like this but are still called KBS. Many KBS may use rule-based programming, but not every rule-based program is a KBS. Equally, it is possible to construct a KBS using a conventional programming language such as C or C++.

3. Technology

The distinction of what is (or is not) a KBS has become blurred due to advances in the technology and the marketing position of the technology providers. For example, when AION and AI Corp (both providers of KBS shells) combined to form the company Trinzic, the press release said nothing about "expert systems" or "KBS". One of the leading PC-based KBS products, Level 5 Object, includes documentation that rarely mentions "knowledge", and never mentions "expert". Similarly, the popular European product Crystal was originally marketed as an "expert system building tool" but was later re-positioned as a rule-based fourth-generation language.

This disassociation with the past is being driven by the requirement to provide generic client/server tools for business applications. Level 5 Object, for example, is marketed as a generic "intelligent" Windows development tool, providing SQL, hyper-media and graphics support in addition to rule-based and object-oriented programming facilities. Complex information and decision support systems are increasingly *hybrid*, employing relational database technology, graphics, knowledge-based components, and quantitative models. What label is attached to the system is generally a function of the visibility of the label in the particular user organization. What used to be called DSS or KBS are now often called "work group systems" or "executive support systems".

3.1 KBS Shells

What used to be referred to as KBS shells therefore tend to be more varied pieces of client/server software. There has been a fall out in the market, to the extent that in the US the PC-based market is dominated by M.4, Level 5 Object and ProKappa. In Europe, Crystal is probably the only PC-based tool that has the market recognition of the American three. For UNIX platforms, NEXPERT OBJECT is a clear market leader. For those needing IBM mainframe and COBOL/CICS compatible shells, the AION Development System and KBMS, from the companies that merged to form Trinzic, are still popular. (Integration with existing data processing systems, particularly so called "legacy systems", has become easier to accomplish. See Reynolds and Beck (1993) for an example of a KBS that is integrated with COBOL, CICS and DB2.) For those wanting powerful, flexible LISP-based tools, then Inference's ART and the tools from the Carnegie-Group are available. There are also a number of specialized tools in areas such as real-time control, machine diagnosis, and help-desk support.

Hayes-Roth and Jacobstein (1994) provide a recent review of KBS technology. The combination of rule-based knowledge supported by hierarchically arranged objects and mixed modes of inference has become standard. The use of meta-rules, which allow for the control and ordering of rules, is also widespread. Support for the construction of hybrid applications is provided in most tools by means of an integrated procedural or functional language (often based on LISP or Prolog), and easy linkage to external modules written using a general-purpose programming language (most often C/C++). Thus, KBS components can readily be harnessed within heterogeneous and distributed software systems.

3.2 New Technologies

Artificial Intelligence (AI) is still generating new usable technologies, some of which take on a life of their own. The machine learning community has contributed several techniques

which have proven effective in synthesising solutions in suitable domains. The best-known of these is probably artificial neural networks (ANN); recently, however, inductive logic programming has achieved notable success.

ANN allow for the construction of connectionist models that develop their own weights from training sets. For a good example of an ANN application, see Tam's work (Tam, 1992, Tam and Kiang, 1992) on the prediction of bank failures; this is a domain in which many have tried to apply rule-based approaches, with very mixed results. Perhaps ANN can best be viewed as alternatives to statistical approaches such as regression and discriminant analysis; in general, ANN cannot be considered to be *knowledge-based* (unless one argues that all knowledge can be reduced to weighted and dynamic links between synapses). However, ANN can effectively be used in conjunction with a KBS in a hybrid system where, for example, certain data used as input to the KBS are pre-classified by a front-end ANN.

Inductive logic programming is more akin to "traditional" KBS technology in that it deals with symbolic knowledge (in the form of logical implications). It is used in domains where, although experts can specify an approximate KB and background knowledge, they are unable to produce a KB which correctly handles a sufficient set of cases. Therefore, the approximate KB are given to an inductive learning system, together with the background knowledge and representative set of example cases, and the learning system synthesises an enhanced KB to classify the required cases. A major advantage over ANN is that the synthesised KB is comprehensible to the experts. Inductive logic programming has been applied with considerable success in domains such as drug design, protein structure prediction, and satellite fault diagnosis (Muggleton, 1992); in one case an inductive logic programming system made what was regarded as a genuine scientific discovery (Sternberg at al., 1992).

In addition to techniques from machine learning, advances in knowledge representation and search methods have opened-up new capabilities for KBS. Case-based reasoning (CBR) offers

a combination of intuitive knowledge representation and analogical-like reasoning, as discussed below. Some of the recent advances in search techniques, especially constraint-satisfaction problem-solving (CSP), can be viewed as alternative equation-solving procedures, often direct alternatives to integer linear programming. We would argue that genetic algorithms can be viewed similarly, since they are essentially concerned with the generation and testing of successive, converging solutions.

CBR tools allow a developer to model a problem situation as (a) a set of previous case, containing observed attributes and the outcome, (b) heuristics that match the present problem to the closest case, and (c) heuristics that solve the problem based upon the best match and the differences with the present problem. They appear to work well where a large number of cases are collected, such as when solving technical problems through help desks. Nguyen et al. (1993) is an interesting application, which also shows the links being made with hyper-media and graphics.

4. Applications

Most discussions on KBS applications focus on the domains that KBS have been used in. It appears that the application of KBS has mirrored overall investments in technology; since 80% of technology investment is in service industries, this is where the bulk of KBS applications are seen. Hayes-Roth and Jacobstein (1994) review the domains of application at the Innovative Applications of AI conference, finding that accounting, finance and other services dominate.

Recent experience (O'Keefe and Rebne, 1993) suggests that a discussion based upon the tasks that the KBS performs is more helpful than one based upon domains. The job of a KBS is to replace or augment a decision making task, and success is often dependent upon understanding of that task, its role in relation to other tasks, and its integration with other

tasks (both manual and automated). There is some evidence that task differences account for differences in the way in which a KBS is managed and operated (O'Keefe et al, 1993).

There are five types of task that appear to be particularly successful. (Those with other experiences could probably generate other tasks.) They are summarized in Table 1.

First, *cumulative-hurdle decision making* (O'Keefe and Rebne, 1993). This is where a number of decisions are made linearly, but the problem may be solvable without overcoming every decision making hurdle. A good example here (which has been a very successful area for KBS) is loan approval. A KBS can handle the first hurdle — logical consistency of the loan application, basic credit worthiness, etc. When a "reject" decision (or, perhaps, an obvious "grant" decision) can not be made, the application can be passed over to an expert for further review. This separation of routine and more complex decision making can be very beneficial when the number of routine problems is a large proportion of the volume, say 80% or more, and the cost of the complex decision making is expensive.

Second, *advisory systems*. These are systems which give simple advice to someone performing a task, such as machine repair, collection of audit data, performing statistical experiments, etc. They are typically applicable to any well defined task that requires the necessary expertise, and usable by a number of different users. They are beneficial when knowledge has to be distributed to professionals due to changes in the law, redesign of machinery, etc. ExperTAX (Shpilberg and Graham, 1987) is probably the classic example.

Third, *heuristic systems*. They typically use heuristics to produce solutions that could be generated from mathematical models, but not in a reasonable amount of time or in a robust manner (e.g., infeasibility can not be easily identified). As might be expected, they tend to be more quantitative than other KBS, and often appear in the same domains as OR models, for example, production scheduling. Duchessi and O'Keefe (1990) presents a production planning KBS where the equivalent ILP could not be solved efficiently. They are especially

successful in domains where imperfect, approximate solutions are of value. Unsurprisingly, this is an area where methods such as ANN and genetic algorithms are having an impact. Success has also been achieved using constraint-based reasoning to solve problems which can be posed in terms of a set of variables and constaints: the problem is expressed in reasonably high-level, declarative terms, and the general constraint-satisfaction mechanism searches for solutions (Freuder & Mackworth, 1994).

Fourth, *configuration systems*. They take a requirement for a configured assembled product (such as a computer, or an air conditioner) and generate the parts needed to configure the product with associated assembly instructions. Following the success of XCON, many engineering companies now have configuration systems, allowing them to offer ˆ *la carte* manufacturing where the customer can define parts of what they are purchasing. The major benefit of these systems is the ability to collapse the order processing cycle, such that an order can be configured and specified for manufacturing within hours, rather than days or weeks. KBS solutions to this kind of problem have been seen to be more reliable than manual solutions (Giarratano and Riley, 1989).

Fifth, *critiquing systems*, sometimes called *expert critics*. These produce critiques of a design or plan that has been produced by a user. They can either be activated by the user as required, or can run in the background, effectively "looking over the shoulder", monitoring the user's actions and suggesting changes when user actions appear to be different from what the critic would do. Silverman and Mezher (1992) provide an introduction, an example, and discuss some research needs. The benefit of these systems is that they can help less experienced users produce better results, and perhaps provide a fail-safe mechanism that can interrupt very poor quality actions.

4.1 Decision Support

What is remarkable about the above list, and successful applications of KBS in general, is that decision support *per se* has not been a successful area. The image of an executive or professional interacting with a knowledge-based DSS to solve problems has not been realized. An executive may use an advisory system, but this only supports a task within the whole decision making framework. Similarly, critiquing and heuristic systems are liable to be part of larger information system (IS) efforts, subservient to (say) design with a CAD tool or producing work schedules in an MRP system. (In fact, some CAD tools are rapidly including advisory and critiquing components that support the design process; the resulting buzz-phrase is "Intelligent CAD", or ICAD.)

The idea that KBS provide a different and better way to develop DSS is at best fanciful (e.g., El-Najdawi and Stylianou, 1993), at worst limiting (e.g., Edwards, 1992). This is for two very important reasons. First, good DSS are intrinsically linked with the requirements of a single or small group of decision makers, to the extent that the interface and design of the system is tailored to their requirements. Taking advantage of codified knowledge requires that it be distributed beyond its originator and original users (most obviously with advisory and configuration systems). At best, the knowledge is automatically applied by software without a user present (many cumulative-hurdle systems are implemented like this — the user only sees problems the KBS can not solve).

Second, with most DSS the user has control over the decision making processes — what information is gathered, how it is presented, the models used, etc. With KBS, this is naturally under the control of the software. KBS that have tried to impose a decision structure on users have often failed — users try to figure out a way around it, so that they can manage problems and decisions as they desire, not as the software dictates. (For an example of this in the context of a generic financial planning KBS, see Sviokla, 1990). A number of companies that have tried to develop and sell generic knowledge-based DSS tools (such as Palladian Software and their Advisor systems (Reitman, 1990)) are out of business.

4.2 Value in Tasks: Business Process Re-engineering and Knowledge-Based Software Engineering

This focus on tasks and how they inter-relate has resulted in KBS becoming an important enabling technology for business process re-engineering (BPR) and knowledge-based software engineering (KBSE). The central realisation in BPR and KBSE is that the most valuable knowledge possessed by an organization lies in key tasks and the processes that use those tasks. This realisation is due to the value-added output derived from performing the processes.

Davenport (1992) discusses a number of BPR projects that use KBS, and many AI conferences presently include sessions on BPR. This link with BPR has given a considerable lift to interest in KBS. Whether BPR is a fad or not is irrelevant; what is important is the realization that (often small) amounts of knowledge can increase the speed, consistency and quality of important business processes. (The fact that the phrase has captured the imagination of management is also useful.) The same is true of KBSE, wherein the processes of building software artefacts are defined by knowledge of tasks, which themselves use certain kinds of knowledge (Lowry, 1992). The processes and tasks can be embodied in tools, such as Celite Corporation and Andersen Consulting's KBDA (Burton, Swanson and Leonard, 1993), which lend structure to the types of knowledge demanded by different applications. These tools and types of knowledge are highly reusable.

BPR and KBSE demonstrate that application of KBS is no longer technology-driven, where proponents hunt do-able applications irrespective of their importance to business.

5. Development Life Cycles

From the outset, KBS development has been associated with prototyping. In some organizations systems are labelled as "KBS" just so that a less formal development methodology can be used; prototyping seems to be acceptable for KBS, but not necessarily for other systems.

There is still some confusion over what prototyping actually is, and the various forms it can take. According to Jojo and O'Keefe (1994),

> "*rapid prototyping* is usually applied when the developer discards earlier versions of a prototype as it becomes obsolete and begins anew. In *incremental prototyping*, the system is broken down into parts, each of which is prototyped and then included one by one in the finished system. In *evolutionary prototyping*, the system is refined over and over again until it satisfies the user's requirements."

Figure 1 shows a fairly typical prototyping oriented development methodology.

Most KBS prototyping is evolutionary — the developer chooses a software tool early in the development process, and the system evolves over time. Knowledge and code may be disposed of or re-coded, but the basic structure of the system and the chosen tool remain. Despite concerns over the management of prototyping, the study in Jojo and O'Keefe (1994) shows that some of these concerns are unwarranted. For example, under-design or limited functionality are often remarked concerns, since the lack of iterations in the development process may limit the ability of the developer to meet user requirements. However, Jojo and O'Keefe (1994), investigating four systems developed at General Electric Industrial and Power Systems, showed that *over-design* was prevalent — the prototyping tended to iteratively cycle while the expert could generate new functionality or until the budget was spent, irrespective of perceived requirements.

Research has shown that for data intensive applications, the combination of some specification plus prototyping works better than prototyping alone. For example, Alavi and Wetherbe (1991) report the results of an experiment where data modelling plus prototyping produced better systems than prototyping alone. We have found similar results for KBS (Lee and O'Keefe, 1994b). In an experimental study, the combination of object hierarchy diagrams, object-based representation and subsequent prototyping was more robust than other development methods, in the sense that functionality, usability and the internal quality of the system were generally better than with other development approaches in the experiment. However, rapid prototyping with just rule-based representation produced the best systems from the perspective of functionality. Thus, rapid prototyping is probably still very useful where quickly uncovering knowledge is important.

Despite the acceptance of prototyping development methods, at least for KBS, problems do persist with large projects. Organizations typically require that sizeable projects staffed by more than one developer have itemized budgets, time tables, and deliverables. There are limited opportunities to obtain a budget of over, say, $50,000 and "run with it." To overcome this, many now see prototyping of a first system as a precursor to a more formalized development process that is initiated after the first system has been implemented and proved some of the benefits that are obtainable from the KBS approach. Figure 2 shows a fairly typical scenario — an idea for application leads to development of a relatively small KBS (typically less than 500 rules and objects, running on a PC), which gets implemented and used. Over time, one of three things happens:

1. The KBS is *used* exactly as intended performing or supporting a task, and provides a small but valuable benefit. Interest in extending its use or functionality is limited, although ongoing maintenance may demand expansion or revision akin to the 3rd possibility below.

2. The KBS *dies*. This can happen early on due to unresolved technical or functionality problems (e.g., interface problems, or inability to deal with important variations of the task undertaken). Alternatively, it can happen after some period of use due to the task becoming irrelevant, or nature of the task changing, such that the benefit of the KBS becomes negligible.

3. The KBS is very successful, and there is a *realization* that its scope can be expanded and *generalized*. For example, the small system that schedules one product can be generalized to schedule many products; the loan approval system can be generalized to other financial product areas.

When realization occurs, subsequent development is liable to be more formal, since the generalized KBS will be a number of times larger than the first KBS. System maintenance is a special case of this kind of generalization; for example, as the R1/XCON application achieved initial success, the need for revision and expansion of the system lead to extensive re-engineering of the KB (van de Brug, Bachant and McDermott, 1986). Often, integration with data management facilities becomes vital when associated data management requirements rise with large systems since the KBS may have to reside externally to where the task is performed. (With small KBS, the KBS can reside locally, and some data requirements can be met by local data entry.)

Due to the need to better integrate a larger system with existing data and software, and that the problem will be better understood due to the initial system, generalized systems are often developed with conventional software. C++ is a special favourite, since any developed objects can be directly expressed in the object-oriented facilities of C++.

5.1 Generalization of Specific KBS

An interesting point here is that long term development of the type outlined in Figure 2 can occur since the knowledge developed in the first small KBS can be either directly used in the larger KBS, or generalized. The most obvious form of generalization is to add more knowledge to the KB in order to broaden the problem-solving ability of the system in its domain. Going further, the existing knowledge may be generalised to solve a broader class of problems within the same domain. For example, a rule developed for scheduling of a specific production line, e.g.

```
if      workload at cell number 14 exceeds present specified threshold
then    send no more work to this cell
```

might be generalizable to

```
if      workload at presently identified bottleneck cell exceeds
        present specified threshold for all type one manufacturing cells
then    send no more work to this cell
```

Thus, to the surprise of some commentators, some KBS have become more general and thus more applicable than originally thought (Duchessi and O'Keefe, forthcoming). IBM's Logistics Management System (Sullivan and Fordyce, 1990), which schedules work in semi-conductor manufacturing, is a particularly good example. Originally developed for one manufacturing line, it was generalized to every line in the plant, then to other plants, and is now available as a generic tool.

Perhaps the most extreme form of generalization is that of abstracting out the task-related knowledge to allow the system to be applied to new domains with the same tasks. There is a long history of this in the KBS field: many of the earliest expert system shells came about as a result of taking a successful task/inference structure and its associated knowledge representation, "emptying out" the specific domain knowledge, and selling the result as a

generic shell. The classic example is EMYCIN ("Empty MYCIN") which, while often being regarded as a general-purpose tool, can best be seen as a generalised task-mechanism for classification applications. Many of the aforementioned specialized KBS tools for task-based domains such as real-time control, machine diagnosis, and help-desk support have similar origins to EMYCIN. We see that generalization, in addition to extending the life of a particular successful KBS application, often results in new "spin off" products.

5.2 Knowledge Acquisition

At this juncture, it is useful to consider the role and placement of knowledge acquisition (KA) in development. KA is not given much coverage here, mainly because it is probably the most extensively researched and developed area of KBS development. Others have done better justice to it than we can. Our experience suggests that KA is generally do-able, but can be time consuming and frustrating. Interestingly, this is often due to the common problems associated with interacting with busy people (such as trying to schedule interviews), rather than any inherent difficulty in acquiring the knowledge. From the perspective here, we will merely highlight a few issues.

In the early stages of development, the tendency is to focus on KA often to the exclusion of all else. Acquiring rules (or whatever) seems to be tantalizingly productive, similar to programming lines of code. But in the same way that generating code that does unnecessary things is ultimately counter-productive, so is acquiring and codifying knowledge that will not be used. Talking to experts or codifying written knowledge to the exclusion of talking to potential users, understanding the task required of the KBS, and assessing its possible contribution to the organization is not recommended. In our experience, the construction of a satisfactory technological artefact with an acceptable quality of knowledge and reasoning ability is merely one factor in ensuring the ultimate success of a KBS project (Preece, 1990). During KA, it is important to consider the entire human-computer system, including its

organizational context, because this will usually have a great impact on the knowledge and tasks which must be built-into the system (Wielinga, Schreiber and Breuker, 1992).

There is a growing move (at least in Europe) towards the use of knowledge modeling methods to document and design KBS. KADS, resulting from an ESPRIT funded project, is probably the best known methodology (for those unfamiliar with KADS, Kingston (1992) gives a very neat account of the use of the basic ideas in a project). These have the advantage of explicitly forcing the developer to model and document the KBS at the knowledge level, rather than the implementation level. The resulting *knowledge-level model* (sometimes called a "conceptual model", somewhat confusingly for those also familiar with Soft Systems Methodology) is intended to allow analysis of the knowledge before a commitment is made to a particular knowledge representation for implementation.

The KADS methodology provides a multi-layer model for knowledge-level modelling (Wielinga, Schreiber and Breuker, 1992). The first layer, the *domain layer*, defines the static objects (facts and relations) of the application domain. The second layer, the *inference layer*, defines the primitive inference steps which can be performed on the objects in the domain layer to derive new objects. The third layer, the *task layer*, defines how to put the primitive inference steps together to solve problems[1]. KADS offers both informal and formal languages with which to describe these layers; normally, they will be described informally at first, and then formalised and refined into an implementation.

Given the comments above that some specification prior to prototyping appears to be important, then use of knowledge modelling is advantageous. A disadvantage is the overhead incurred in maintaining the knowledge model — this can slow development, and may be time consuming when requirements are rapidly changing and the scope of KA

[1]Earlier descriptions of KADS identify a fourth layer, the strategy layer, which defines knowledge for selecting between different tasks to solve a problem.

similarly shifting. There is recognition of this problem among proponents of the knowledge modelling approach, some of whom are developing tools which assist in rapidly turning KADS models into executable system prototypes (Anjeweirden, Wielemaker and Toussaint, 1992).

5.3 Formal Specification for KBS

One present view of KBS, which is gaining support, is that KBS development requires a large dose of formal methods. A system should have formalized requirements, which lead to a system specification, which can then be implemented using formal programming languages, or proved using formal methods (Gamble, 1991; Gold and Plant, 1994). There are at least two factors explaining the growth in interest in formal methods. The first is a "technology push": use of various flavours of formal logic in both knowledge representation languages (most obviously Prolog) and formal specification languages for conventional software (for example, Z and VDM) suggests — perhaps seductively — a convergence of approaches. There is also an "application pull": many of the advocates of formal approaches to KBS development are concerned with the development of KBS in safety-critical domains, such as the aerospace and nuclear industries.

There does seem to be a crucial difference between KBS and conventional software in this respect, however. For conventional software, formal methods are used to create an initial specification of requirements, which is then refined through successive levels of detail to an implementation. Correctness of the implementation is ensured in a similar way to a mathematical proof: the initial specification is taken as correct, and each refinement step follows well-defined rules. The initial specification can be taken as correct because the users' requirements are well-understood. This is often not true for KBS applications, however, particularly those in domains lacking a formal foundational theory, such as the medical, legal, and business domains. In these domains, a *correct* formal specification of the users'

requirements will most likely be very hard to create, and a *complete* specification may be impossible to create (Bellman, 1990; Gamble, 1993).

Therefore, from the point of view of those working with business systems, increased formality is not attractive (O'Keefe and O'Leary, 1993). KBS development is attractive to many in business *because* of the lack of formality — the ability to investigate and change requirements, and for these to evolve over time. Freezing the scope of a project too early can result in disaster. With KBS, we are investigating problems and tasks that we do not understand very well, at least at the outset. If we did understand them, we would not be using KBS — likely we would be programming an algorithm, or using an off-the-shelf solution.

6. Verification and Validation (V&V)

According to the old adage, "verification is building the system right, validation is building the right system" (O'Keefe, Balci and Smith, 1987). Thus verification is aimed at eliminating errors in the system, and is typically a software and programming task. Validation is more concerned with the quality of the system; the extent to which it performs its task, the degree of accuracy, and the observed robustness. Implicitly, verification is part of validation: a system that is not "built right" is unlikely to be "the right system".

6.1 Verification of KBS

Verification of KBS has focused on attempts to demonstrate that a KBS has been built in conformance with a number of well-defined properties, chiefly freedom from logical conflict, redundancy, and deficiency (Preece, Shinghal and Batarekh, 1992). *Conflict* refers to the ability of the system to arrive at logically-inconsistent conclusions from consistent input.

Redundancy refers to the presence within the system of logically-unnecessary structures which never affect the relationship between the input and output of the system. *Deficiency* refers to the absence of structures which should be present, logically, for the system to arrive at conclusions for all valid input cases.

In practice, it is difficult to prove the absence of conflict, redundancy and deficiency in a KBS built using the kind of tools described earlier. This is because, in order to provide the rich functionality demanded by programmers, all current industrial-strength KBS tools offer extra-logical features which defy a pure logical analysis. To work around this problem, KBS verification tools look instead for *anomalies*, where an anomaly is an *apparent* instance of conflict, redundancy, or deficiency, based upon a logical analysis of the KBS. Thus an anomaly is a potential error — it may be an actual error that needs fixing, something that causes efficiency or maintenance problems, or it may be harmless (or even intentional). Anomaly detection procedures have focused on rules, since rules have dominated KBS development, and there are a large number of categorized anomalies and some tools to detect them (Preece, Shinghal and Batarekh, 1992; O'Keefe and O'Leary, 1993). Examples of the use of one such tool on realistic KBS applications are provided in Preece and Shinghal (1994).

Recently, the notion of KBS verification has expanded to reflect the use of formal specification languages (Treur and Wetter, 1993). As discussed earlier, these languages facilitate the creation of formal descriptions of KBS, which are amenable to formal proof techniques, and from which implementations can be derived that conform to the specifications. However, because of the difficulty in creating such specifications for KBS with ill-defined user requirements, this approach to verification has met with very limited success to date (Gamble, 1993; Rushby and Whitehurst, 1989).

6.2 Validation of KBS

From the perspective of this paper, validation overshadows formal verification. In an ideal world, a verified KBS would be a naturally-validated KBS (Adrion, 1982), but this is far from what is currently possible in practice, especially in business applications of KBS. Even if it were possible to specify formally all of the users' requirements, and then to verify that an implementation conforms to this specification, there would still be no guarantee that the requirements were correct. Therefore, it seems that validation must always be more than verification.

Validation bridges the gap between performance of the KBS (the software divorced from setting and user), and performance of the system (the software in place being used to perform its task). Table 2 presents a number of methods for validation taken from O'Keefe and O'Leary (1993); helpfully it is somewhat self-explanatory. Validators typically use methods to investigate the components of a KBS (individual rules, programmed heuristics, the knowledge or conceptual model), and the entire KBS itself.

A KBS must have *criteria* against which it can be validated. Criteria based upon performance, accuracy, and user-oriented attributes (such as ease-of-use) are common, and it is unlikely that a single criterion will suffice. (Adelman, 1992, discusses this in great detail, and is required reading for anyone validating and evaluating a KBS.) Most validation methods are applicable to performance related criteria, especially *acceptable level of performance* — the notional level of performance that the KBS must perform at to be a useful surrogate or supporting system for decision making. This is very task and application dependent, and thus validation for a specific application does not validate a KBS as usable for other tasks. There is no absolute stamp of validity, as there might be with (for instance) a programming language compiler or database system.

By far the most prevalent method of validation is case testing, where the validator runs cases previously solved by an expert or experts through the system, and compares performance of the system to the experts. This benchmarking approach is very convenient, and has the

advantage that statistical techniques may be employed to make the comparison sounder. However, in practice it is not always easy to implement. It requires good cases that match those that the system will encounter. Where attributes of the problem have changed significantly, or it is now recognized that previously highly regarded solutions are not necessarily that good (think about advances in medicine), the approach may not be that good. It also assumes that the experts are the targets for performance, but many validators find that expert performance is mixed and often contradictory (think about the performance of financial advisors).

There has been some consideration of the criteria for what constitutes a good set of test cases for a KBS. In general, it is not possible to create a set of test cases which is guaranteed to expose all faults in an arbitrarily-complex piece of software, including all KBS (Goodenough and Gerhart, 1975). Instead, it is necessary to create test cases to show that known types of error are *not* present. In conventional software testing, errors of *commission* can be exposed through structural testing criteria, aimed at testing execution paths through the program code. Analogous methods can be employed for KBS, provided that a workable notion of execution path can be developed (Grossner et al, 1993; Kiper, 1992). Errors of *omission* can be exposed through functional testing criteria, based upon specifications of the users' requirements. This approach does not necessarily require complete formal specifications, since classes of potential functional faults can be identified in practice for KBS by rigorous engineering approaches (Miller, 1990; Rushby, 1988). Limited experiments performed to date — albeit using somewhat simple application systems — have suggested that a combination of structural and functional testing is a highly effective means of exposing faults in a KBS (Kirani, Zualkernan and Tsai, 1992; Rushby and Crow, 1990).

It is worth noting in Table 2 that most validation methods — including the testing approaches — are applicable to the later stages of the life cycle when the system has achieved some level of performance. This often results in validation being driven out of development due to budgetary problems and the pressure to fully implement. How much a system is

under-validated tends to be a function of its task; developers are not willing to let life, money or mission critical KBS be fielded and fail, but typically take a far more liberal view when a system is not critical.

6.3 Validation Strategies

Many KBS projects are driven by KA, where what knowledge is required next acts as a driving force determining the next step in the project. It may be more sensible, however, for a project to be validation driven, where the gap between present and desired performance drives the project. Lee and O'Keefe (1994a) discuss this, and in the context of a KBS project present some guidelines for generating a *strategy* that specifies the validation methods and their timing. The method assumes that the strategy will be liable to changes based upon what is learnt as the project progresses.

To generate a strategy, some guidelines are:

1. *Choose and specify the criteria for V&V*. This should be done in as much detail as possible. Although this can be changed 'down the line', and for some systems may not become clear until some KA is performed, having a clear as possible goal for V&V will greatly aid development. In addition to performance related criteria, where usage is optional, usage-oriented criteria should also be developed early on, even if how to measure for usage-oriented validity may not be immediately apparent.

2. *Develop a list of suitable V&V methods given the intended characteristics of the KBS and the identified criteria*. This requires some familiarity with the methods, including those shown in Table 2, and where they are best applied.

3. *Map the V&V methods onto the life cycle being used*. Specify *where* the methods will be used and *when*. This mapping can be revised, but revision should not mean frequently postponing V&V until the later stages of the life cycle (where issues of cost, timing and resource availability will come into play).

This may appear to be little more than common sense. Our experience, however, suggests that few developers enter a project with any preconceived ideas about how to do validation.

7. Implementation

KBS rarely fail due to technical problems. Despite the advanced nature of some of the technology, developers appear to be able to work with it, and when necessary, overcome some of its limitations with novel procedures and fixes.

Implementing KBS into organizations is far from trivial (Sharma, Conrath and Dilts, 1991). Although this is probably a truism for all but the simplest of Information Systems, it appears that successful implementation of KBS is sometimes more difficult than for other systems. This is due (at least in part) to the task-oriented focus of KBS — the success of the KBS is related to the successful management and importance of the task as much as the successful performance of the KBS.

In a study on the success of KBS in three organizations, Duchessi and O'Keefe (1992, forthcoming) found some common factors among successful systems. Two of these, namely management commitment to the project and the importance of user participation, are common to successful IS in general. More interestingly, it was found that:

1. *Systems must have demonstrable benefits*. It is not sufficient for a KBS to be found to be "usable" or even "useful". It must make a measurable contribution to the organization,

such as a decrease in down-time or cycle-time, or an increase in market share. If the outcome from a task is measured (*ˆ la* re-engineering), then a KBS should have an impact on those measures.

2. *There must be growth in the underlying application area*. The underlying application (or task) must be growing in either importance or frequency or execution; thus a KBS can contribute to that growth. Everyone remembers XCON due to the success of the VAX computer; no one can name the KBS developed to support the configuration of now discontinued computer products.

3. *There is a need to control and understand data*. Building a KBS, and then expecting the data required to use it to magically appear, is an overture to failure. It is necessary to understand the available data, and perhaps even build some data capture and management facilities. Not only is this necessary to support the implemented KBS, but understanding the data can help determine the scope and role of the KBS.

Regarding behavioural issues, implementing a KBS can have profound effects on the way that knowledge is distributed in an organization. Given the association with specific tasks, workers performing those tasks can find their job enhanced, devalued or even replaced. The announcement made by AT&T that they would replace the majority of their phone operators with a speech recognition system is an extreme case, and perhaps the first reported large-scale displacement of workers by AI (Duchessi, O'Keefe and O'Leary, 1993).

8. Conclusions

So, for an Operational Researcher (or other person working on business and managerial problems and systems) where does KBS fit in? Are they another technique, or a complete philosophy?

Our personal opinion is that KBS is best viewed as another set of modelling techniques and methods, with associated software and system development approaches. Whereas mathematical programming and discrete-event simulation are both considered as OR techniques, in truth they are now individual well developed schools of thought, each with their own conferences, practitioners and software providers. KBS is similar, but generally not considered part of OR (the way that mathematical programming and simulation are). This is simply because OR people did not discover and propagate the ideas. From an OR perspective (especially pedagogical), the fact that KBS is *out* and mathematical programming *in* is a quirk of history.

Thus we return to the paragraph quoted at the start of the paper. One possible rewrite is:

> "We hope that the development and implementation of KBS is seen as a useful technology, readily available via software. As such, KBS should be considered as similar to packaged modeling methods such as linear programming and discrete-event simulation, but not as fundamental a tool as statistics or basic computing skills. Evidence suggests that for certain types of tasks, KBS is a viable method for automating, augmenting or supporting the task. KBS should not be defined as a decision support method *per se*, since the applications of KBS are more multi-faceted than simple support and advisory roles. Further, efforts to build large-scale generic DSS based upon KBS method have not been successful; KBS is perhaps best seen as a possible building block in DSS rather than the cement."

Of course, other rewrites (using better grammar and metaphors, let alone more insight) are possible.

Acknowledgements

Our understanding of KBS is dependent upon fellow colleagues and researchers over recent years. Bob O'Keefe would particularly like to acknowledge the insights acquired from working with Val Belton (University of Strathclyde, Scotland), Peter Duchessi (State University on New York at Albany, USA), Dan O'Leary (University of Southern California, USA), Sunro Lee (Hong Kong University of Science and Technology) and Walter Reitman (Rensselaer Polytechnic Institute, USA). Alun Preece would like to thank former colleagues in the Computer Science Department/CENPARMI, Concordia University, MontrŽal, Canada, the members of LIA, UniversitŽ de Savoie, France, and Laurie Moseley (University College Swansea, Wales).

References

L. Adelman (1992) *Evaluating Decision Support and Expert Systems*, John Wiley & Sons, New York, NY.

W.R. Adrion, M.A. Branstad and J.C. Cherniavsky (1982) Validation, verification and testing of computer software, *ACM Computing Surveys*. 2(June), 159-192.

M. Alavi and J.C. Wetherbe (1991) Mixing prototyping and data modeling for information system design, *IEEE Software*, 8(May), 86-91.

A. Anjeweirden, J. Wielemaker and C. Toussaint (1992) Shelley: computer-aided knowledge engineering, *Knowledge Acquisition* 4:1, 109-126.

V. Barker and D. O'Connor (1989) Expert systems for configuration at Digital, XCON and beyond, *Communications of the ACM* 32:3, 298-320.

K. Bellman (1990) The modeling issues inherent in testing and evaluating knowledge-based systems, *Expert Systems with Applications*, 1:3, 199-215.

A. van de Brug, J. Bachant and J. McDermott (1986) The Taming of R1, *IEEE Expert* 1:3, 33-39.

S. Burton, K. Swanson and L. Leonard (1993) Quality and knowledge in Software Engineering, *AI Magazine* 14:4, 43-50.

T.H. Davenport (1993) *Process Improvement: Reengineering Work Through Information Technology*, Harvard Business School Press, Boston, MA.

P. Duchessi and R.M. O'Keefe (1990) A knowledge-based approach to production planning, *Journal of the Operational Research Society* 41:5, 377-390.

P. Duchessi and R.M. O'Keefe (1992) Contrasting successful and unsuccessful expert systems, *European Journal of Operational Research* 61:1/2, 122-134.

P. Duchessi and R.M. O'Keefe, Evolutionary steps in expert systems projects, forthcoming in *Expert Systems With Applications*.

P. Duchessi, R.M. O'Keefe and D.E. O'Leary (1993) A research perspective: artificial intelligence, organizations and management, International Journal of *Intelligent Systems in Accounting, Finance and Management* 2:3, 151-159.

J.S. Edwards (1992) Expert systems in management and administration — are they really different from decision support systems?, *European Journal of Operational Research* 61:1/2, 114-121.

M.K. El-Najdawi and A.C. Stylianou (1993) Expert support systems: Integrating AI technologies, *Communications of the ACM* 36:12, 54-65.

E.C. Freuder & A.K. Mackworth (1994) Constraint-Based Reasoning. MIT Press, Cambridge MA.

R.F. Gamble (1993) A perspective on formal verification, in A.D. Preece, ed., *Validation and Verification of Knowledge-Based Systems (AAAI-93 Workshop Notes),* AAAI Press, 131-134.

R.F. Gamble, G.-C. Roman and W.E. Ball (1991) Formal verification of pure production system programs, in *Proceedings 9th National Conference on Artificial Intelligence (AAAI 91),* AAAI Press, 329-334.

J. Giarratano and G. Riley (1989) *Expert Systems: Principles and Programming*, PWS-Kent, New York, NY.

D.I. Gold and R.T. Plant (1994) Towards the formal specification of an OPS5 production system architecture, *International Journal of Intelligent Systems*, 9:8, 739-768.

J.B. Goodenough and S.L. Gerhart (1975) Toward a theory of data selection. *IEEE Transactions on Software Engineering*, SE-1:2, 156-173.

C. Grossner, A. Preece, P.G. Chander, T. Radhakrishnan and C.Y. Suen (1993), Exploring the structure of rule based systems, in *Proceedings 11th National Conference on Artificial Intelligence (AAAI 93),* AAAI Press, 704-709.

F. Hayes-Roth and N. Jacobstein (1994) The state of knowledge-based systems, *Communications of the ACM* 37:3, 27-39.

P. Holroyd, G. Mallory, D.H.R. Price and J.A. Sharp (1985) *Developing expert systems for management applications*, Omega 13, 1-11.

L. Jojo and R.M. O'Keefe (1994) Experiences with an expert system prototyping methodology, *Expert Systems: The International Journal of Knowledge Engineering* 11:1, 13-22.

J.K. Kastner and S.J. Hong (1984) A review of expert systems, *European Journal of Operational Research* 18:3, 285-292.

J. Kingston (1992) Pragmatic KADS: a methodological approach to a small knowledge-based systems project, *Expert Systems: The International Journal of Knowledge Engineering*, 9:4, 171-180.

M. Sternberg, R. Lewis, R. King and S. Muggleton (1992) Modelling the structure and function of enzymes by machine learning, in *Proceedings of the Royal Society of Chemistry: Faraday Discussions*, 93, 269-280.

S. Kirani, I. Zualkernan and W.-T. Tsai (1992) *Comparative evaluation of expert system testing methods*, Technical Report TR 92-30, University of Minnesota, Department of Computer Science.

S. Lee and R.M. O'Keefe (1994a) Developing a strategy for expert system verification and validation, *IEEE Transactions on Systems, Man and Cybernetics* 24:4, 643-655.

S. Lee and R.M. O'Keefe (1994b) *An experimental investigation into the determinants of knowledge-based systems quality*, Technical Report, Department of Business Information Systems, Hong Kong University of Science and Technology.

M. Lowry (1992) Software engineering in the twenty-first century, *AI Magazine* 13:3, 71-87.

L.A. Miller (1990) Dynamic testing of knowledge bases using the heuristic testing approach, *Expert Systems with Applications*, 1:3, 249-269.

S. Muggleton (1992) *Inductive Logic Programming*. Academic Press, London.

T. Nguyen, M. Czerwinski and D. Lee (1993) Compaq QUICKSOURCE: Providing the consumer with the power of AI, *AI Magazine* 14:3, 50-60.

R.M. O'Keefe (1985) Expert systems and operational research — mutual benefits, *Journal of the Operational Research Society* 36:2, 125-129.

R.M. O'Keefe, O. Balci and E.P. Smith (1987) Validating expert system performance, *IEEE Expert* 2:4, 81-90.

R.M. O'Keefe, V. Belton and T. Ball (1986) Experiences with using expert systems in O.R., *Journal of the Operational Research Society* 37:7, 657-668.

R.M. O'Keefe and D.E. O'Leary (1993) A review and survey of expert system verification and validation, *Artificial Intelligence Review* 7:1, 3-42.

R.M. O'Keefe and D. Rebne (1993) Understanding the applicability of expert systems, *International Journal of Applied Expert Systems* 1:1, 3-24.

R.M. O'Keefe, D. Rebne, D.E. O'Leary and Q Chung (1993) The impact of expert systems: System characteristics, productivity and work unit effect, *International Journal of Intelligent Systems in Accounting, Finance and Management* 2:3, 177-189.

D.E. O'Leary and R.M. O'Keefe (1993) The use of formal methods for specifying KBS: The case of business systems, presented at the *AAAI Workshop on Verification, Validation and Testing of KBS*.

A.D. Preece, R. Shinghal and A. Batarekh (1992). Principles and practice in verifying rule-based systems, *Knowledge Engineering Review*, 7:2, 115-141.

A. Preece and R. Shinghal (1994) Foundation and application of knowledge base verification, *International Journal of Intelligent Systems*, 9:8, 683-702.

A.D. Preece (1990) Towards a methodology for evaluating expert systems, *Expert Systems*, 7:4, 215-223.

W. Reitman (1990) Generic expert systems for management applications: the Operations Advisor and the Management Advisor, *Computer Science in Economics and Management* 3, 167-175.

D. Reynolds and T. Beck (1993) Tennessee offender management information system, *AI Magazine* 14:3, 61-68.

J. Rushby (1988) *Quality measures and assurance for AI software*, NASA Contractor Report CR-4187, SRI International, Menlo Park CA.

J. Rushby and A. Whitehurst (1989) *Formal verification of AI software*, NASA Contractor Report CR-181827, SRI International, Menlo Park CA.

J. Rushby and J. Crow (1990) *Evaluation of an expert system for fault detection, isolation, and recovery in the manned maneuvering unit*, NASA Contractor Report CR-187466, SRI International, Menlo Park CA.

R.S. Sharma, D.W. Conrath and D.M. Dilts (1991) A Socio-technical model for deploying expert systems — Part I: The general theory, *IEEE Transactions on Engineering Management*, 38:1, 14-23.

D. Shpilberg and L.E. Graham (1989) Developing ExperTAX: An expert system for corporate tax accrual and planning, in M.A. Vasarhelyi (ed.) *Artificial Intelligence in Accounting and Auditing, Markus Weiner*, New York, NY, 343-372.

G. Sullivan and K. Fordyce (1990) IBM Burlington's logistics management system, *Interfaces*, 20:1, 43-64.

J.J. Sviokla (1990) Expert systems and their impact on the firm: the effects of PlanPower use on the information processing capacity of The Financial Collaborative, *Journal of Management Information Systems* 6:3, 65-84.

B.G. Silverman and T.M. Mezhar (1992) Expert critics in engineering design: Lessons learned and research needs, *AI Magazine* 13:1, 45-62.

K.Y. Tam (1992) Neural network models and the prediction of bank bankruptcy, *Omega* 19:5, 429-445.

K.Y. Tam and Y.H. Kiang (1992) Managerial applications of neural networks: the case of bank failure predictions, *Management Science* 38:7, 926-947.

J. Treur and T. Wetter (1993) *Formal specification of complex reasoning systems*, Ellis Horwood, Chichester, England.

B.J. Wielinga, A.Th. Schreiber and J.A. Breuker (1992) KADS: a modelling approach to knowledge engineering, *Knowledge Acquisition* 4:1, 5-54.