KNOWLEDGE BASED SYSTEM DEVELOPMENT TOOLS

John K.C. Kingston
AIAI, University of Edinburgh, Scotland

Keywords: Knowledge based systems, programming tools, Artificial Intelligence

Contents

Summary

Knowledge based system programming tools provide a range for facilities for representing knowledge and reasoning with knowledge. The purpose of these facilities is to allow knowledge based systems to be constructed quickly. This article categorises knowledge based system development tools, supplies examples of developed KBS applications, and discusses the features to consider when selecting a tool for a project.

1. Introduction

Artificial Intelligence tools are almost exclusively software tools; programming languages or program development environments. For Artificial Intelligence is a software-based discipline; despite the popular image of AI research focusing on self-aware robots or autonomous vehicles, hardware problems for artificial intelligence rarely go beyond the integration of motors, gears and video cameras. As the name suggests, the main research topic of Artificial Intelligence is the development of

intelligence, and all existing approaches to modelling, developing, representing or producing intelligence are software-based.

In practice, the development of intelligence - building a machine that can think for itself - has proved difficult. As a result, Artificial Intelligence research has (broadly speaking) divided into two approaches: the "cognitive science" approach to developing intelligence, and the "knowledge engineering" approach to replicating the results of intelligent behaviour. The issue that divides them has been whether an intelligent machine must "think" like humans, or merely use the same knowledge as humans to produce the same results.

The "cognitive science" approach to Artificial Intelligence has attempted to develop intelligent machines by replicating biological mechanisms of human thinking - these "neural network" techniques are generically referred to as a connectionist approach - or by using "genetic mutation" of software in order to develop more 'intelligent' programs or "artificial life", which is usually referred to as an evolutionary approach. It's difficult to determine whether these techniques can produce truly "intelligent machines", largely because of difficulties in defining and measuring intelligence. Dictionary definitions define intelligence in terms of possessing 'understanding' or 'knowledge', but provide no way of measuring these qualities. At a deeper level, the question of what makes a human or a machine "intelligent" is not only difficult to answer, but also emotive; if a machine truly thinks like a human, is it intelligent? Does it deserve human rights? Is it acceptable to manufacture a machine that is more intelligent than the programmer?

Can a machine 'evolve' intelligence without intervention from its creator? The answer to the last question is currently 'no' (all evolutionary algorithms need some guidance, or modifications of input parameters, before they can produce a good result), but the remaining questions are offered as an exercise for the reader; recommended reading might include the works of Isaac Asimov and Arthur C. Clarke.

The proponents of the "knowledge engineering" approach to Artificial Intelligence have chosen to tackle the question of defining intelligence by taking one of the key words from the dictionary definition - 'knowledge' - and choosing to define this instead. In order to do this, they have drawn on several psychological theories of human thinking, as well as on philosophy, linguistics and mathematics. The aim has been to understand what constitutes "expert knowledge" - that is, the knowledge of someone who is highly experienced and highly skilled at performing a particular task - in order to capture that same knowledge and represent it within a computer program, so that given a particular problem to solve, the computer program could produce the same outputs as the expert. The emphasis in the "knowledge engineering" approach has shifted away from replicating the mechanisms of human intelligence to replicating the outputs of intelligent humans. Since knowledge is generally represented within these programs using structured collections of "symbols" (words, concepts, ideas, and values), this approach is sometimes referred to as a *symbolic* approach to Artificial Intelligence.

The distinction between the "cognitive science" and "knowledge engineering" approaches is reflected in the tools that are available to support these approaches. Figure 1 shows a hierarchy that encapsulates the major distinctions between tools. Since the

"knowledge engineering" approach requires considerable effort in collecting and analysing knowledge before it can be programmed, a selection of "knowledge engineering" support tools are also identified.

The purpose of this article is to describe and discuss "knowledge engineering" programming tools - i.e. tools for programming "expert systems" or "knowledge based systems" (KBS). Other articles in this Encyclopedia cover knowledge engineering support tools and the "cognitive science" programming tools. The remainder of this article will discuss differences between tools for KBS programming, with illustrative examples to show the sorts of tasks that each type of tool can tackle.

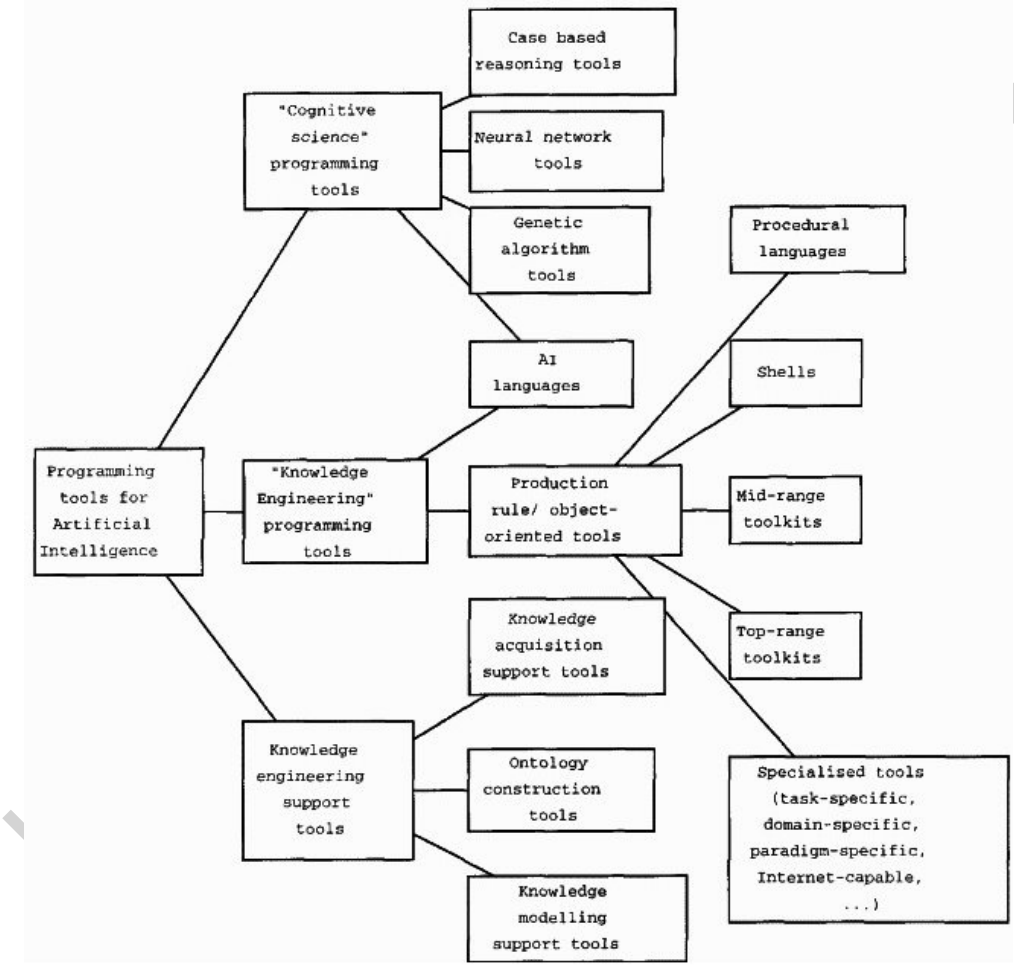

Figure 1: Partial taxonomy of tools for Artificial Intelligence

2. KBS Tools: Functionality

Knowledge based systems consist of three main components: the *knowledge base*, which consists of facts or information about "the world", plus information about permitted inferences on these facts; the *inference engine* which enables inferences to be drawn from the knowledge base to generate new facts and information; and the *user*

interface. For example, a KBS to diagnose hardware faults in computers might contain facts about computers and how they are assembled and details of inferences that can be drawn from certain symptoms (e.g. "If the power light is on, then the electrical supply is not faulty"). The idea behind a KBS tool is that it supplies an inference engine, alongside frameworks for entering facts and information about the world, and (sometimes rudimentary) facilities for building a user interface. A KBS programmer is then only required to input facts and information into the knowledge base, and to specify permitted inferences for the inference engine, in order to construct a knowledge based system.

The inference functionality provided by a KBS tool is the prime determiner of the contents of the knowledge base. KBS tools typically provide one or more of the following programming techniques:

- Production rules (forward chaining);
- Production rules (backward chaining);
- Object oriented programming.

In addition, other inference facilities may be available, including access to functional or procedural programming. The one facility that is present in some KBS tools that is rarely found in any other tools is the ability to perform *hypothetical reasoning.*
Each of these techniques/facilities will be described briefly below.

2.1 Production rules; forward and backward chaining

Production rules are statements of the general form "IF A and B and C are true then D is true and action E must be taken". Their name comes from their capability to "produce" new information from existing information. Within KBS tools, the conditions of the rule (A, B and C in the above example), also called the *left hand side* of the rule, are represented as patterns that can be matched against a "working memory" of facts or objects; if the patterns are matched, the rule can be "fired", allowing the deductions and actions (the *right hand side* of the rule) to be carried out. For example, a rule that restricted nightclub admission to the "young, free and single" might appear as follows in a KBS tool:

(rule young-free-and-single
IF
(object? person (is-a person) (age Page (test (< Page 25))) (marital-status single) (major-commitments none))
THEN
(assert (? person nightclub-admission yes))

The fact that a person has no major commitments may have been deduced by another rule; the fact that the person can be admitted to a nightclub may trigger further rules. This process, where the output of one rule matches and triggers the conditions of another rule, is known as rule *chaining.*

Chaining can occur forwards or backwards; it turns out that the direction of rule chaining can have a big effect on the efficiency of a knowledge based system, so this

distinction is worth describing in detail.

Forward chaining rules start with a set of known data, and try to match the conditions of rules against that data. Several rules may have all their conditions matched; some rules may have their conditions matched in several ways by different combinations of information. In most KBS tools that support forward chaining, one matched rule is then fired, the "working memory" of information is updated, and the matching process takes place once again. This continues until no more rules can fire, or until a sufficiently good solution is found. Forward chaining is a good approach when a lot of data has already been collected. It is also the only feasible approach for synthetic tasks such as planning or configuration problems, where there are a near-infinite number of possible solutions. However, the repeated matching process requires the conditions of rules to be compiled into a network, known as the "RETE network", if the rules are to run at any reasonable speed. This places some restrictions on the conditions that can appear in a production rule in a forward chaining system; for example, attribute names (such as *age* and *marital-status* in the example above) may not appear as variables in a rule.

Backward chaining rules start with a conclusion that needs to be proved, and tries to find existing information that matches that conclusion, or rules that could deduce that conclusion. When suitable rules are found, one such rule is selected, and the conditions of that rule become sub-goals that must now be proved. The system then searches for rules capable of producing these sub-goals. In other words, by performing pattern matching between goals and the right hand side of rules, backward chaining will traverse a 'chain' of rules in the opposite direction to forward chaining. Backward chaining concludes when all the sub-goals and sub-sub-goals of the final conclusion are proved. If it finds no rules (or existing information) to prove a particular sub-goal, it 'backtracks' to the last point at which it had a choice of rules, and chooses another rule.

Backward chaining is especially good where there is not much existing information, and the information has to be gathered by asking the user. Using the example rule given above, backward chaining might ask a person their age; if the answer is over 25, backward chaining would not ask any further questions about marital status or major commitments, unless they were required by another rule. Backward chaining is therefore good at avoiding unnecessary questions.

Backward chaining is also good for tasks where there are many more possible data than conclusions. Diagnostic tasks are a good example of such tasks, and of the many commercial KBS applications that carry out diagnostic tasks, nearly all were developed using backward chaining rules.

2.2 Object oriented programming

Object oriented programming is based on two fundamental assumptions:

- Information about a single "object" in the world should be gathered together in a single data structure;
- It is important that a program preserves the structure of relationships between objects.

Based on these assumptions, an object oriented program consists of a collection of objects with relationships between them; a common relationship is the class-subclass relationship, also known as *is-a*, an *inheritance* relationship, or a *taxonomic* relationship. The programming procedures are all attached to particular objects; for example, a procedure that determined whether someone was "young, free and single" would be attached to the *person* object, and (by inheritance) to every instance of the *person* object (i.e. every person). Procedures are invoked by sending appropriate messages to objects; so if an object representing a nightclub wanted to determine whether a person should be admitted, the "nightclub" object (strictly speaking, an instance of the nightclub object) would send a message to the "person" object, asking it to evaluate whether it was young, free and single. The "person" object would run its "young free and single" procedure and return a yes/no value, allowing the "nightclub" object to proceed with its reasoning about admission.

Object oriented programming is useful where the structure and relationships between objects is important to problem solving (for example, in a classification task).

2.3. Hypothetical reasoning

Hypothetical reasoning requires that certain facts are assumed to be true; that the basis of this assumption is known; and that all further reasoning based on these facts is also known to be assumption-based. The standard approach to assumption-based reasoning in knowledge based systems is to use an "assumption-based truth maintenance system" (ATMS). In practice, this equates to a mechanism whereby "possible worlds" are created; in each "possible world", some facts take on different values. Reasoning can then take place to deduce the consequences of these altered values. Hypothetical reasoning is very useful for implementing problems involving search (which is common in Artificial Intelligence) or problems requiring prediction of future values.

In many KBS tools, "possible worlds" have not been implemented, however, because of memory restrictions. A few KBS tools use an alternative truth maintenance system known as "logical dependency", in which facts are tagged as being dependent on continued existence of other facts. Unfortunately, logical dependency doesn't work well with objects. Logical dependency is useful for supporting systems where a user is asked a series of questions, but the user might want to change an answer to an earlier question.

3. KBS Tools: Classification

In the late 1980s and early 1990s, when commercial interest in knowledge based systems was at its peak, approximately 200 KBS tools were commercially available. Many are still available but are no longer described as KBS tools for marketing reasons; common alternative terms include "intelligent decision support tools", "enterprise support tools" or "knowledge management tools". Close reading of the marketing descriptions often reveals that the tools can support "business logic" or the "implementing of business rules", which appears to be the acceptable description of KBS tools in a business world that considers "artificial intelligence" to be relevant only to the computer games industry. For the purposes of this article, all tools that are capable of being used to develop knowledge based systems - whatever they are called in

their marketing literature - will be considered as "KBS tools".

In order to be able to discuss and describe such a large number of tools, it is necessary to divide them into subcategories; but since these tools differ in many ways, such as inference mechanisms, flexibility, portability, and ease of use, what are the most important distinguishing criteria?

When a project was carried out to develop a KBS to support the selection of KBS tools (see references), knowledge acquisition suggested that the most useful dimensions for classification of KBS tools are price, functionality and required hardware platform. It also appeared that all of these are closely related (based upon the observation that additional functionality usually comes at additional cost, and requires more powerful hardware). Surveys of previous attempts to classify KBS tools suggest that most authors use a broadly similar classification; for example, one author groups the products into three main categories based primarily on functionality, which also happen to differ markedly in the hardware platforms on which they are available. Another author has a similar classification, with the addition of a distinction between toolkits - KBS tools offering multiple programming techniques - with "closely coupled" functionality (where the multiple techniques can be mixed and matched at will) and "loosely coupled" functionality (which consists of a collection of tools or languages with a common repository). While the power of hardware platforms has increased greatly since many of these tools were produced, thus greatly reducing the need for specialized hardware for more powerful toolkits, it is still helpful to classify KBS tools according to these three attributes.

Using these three attributes, KBS tools can be classified into one of the categories described below. N.B. Where particular tools are named, some details about the current vendors of these tools can be found at the end of this article.

3.1. Classifying KBS tools: Shells

Shells are the smallest and simplest of all KBS tools. They were generally designed to run on the PCs available in the late 1980s (8086s or 286s, with kilobytes rather than megabytes of RAM); a few were designed to run on Macintosh PCs rather than IBM-compatible PCs. Their price was comparatively low, typically less than 2000 Euro. They offer only a single technique for programming KBS (usually production rules, chaining forward **or** backward). Their rules often resemble natural language more closely than the rules of other KBS tools: for example, the following is a rule from VP-EXPERT (taken from "VP-EXPERT Examples,
(http://www.cis.ysu.edu/~john/824/ex824.html "):

RULE measles

IF temp_range = very_high AND spots = yes AND inoculated <> yes
THEN diagnosis = measles
DISPLAY "Your very high temperature and spots indicate a case of measles"
BECAUSE "Very high temperature and spots usually indicate measles (unless the patient has already been inoculated for it) "

Shells have been the most widely used tools in KBS development, probably because of the low cost of software and hardware, the ease of learning to program rules such as the one above, and the widespread availability of low-specification PCs over a long period. 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 must contain only a single conclusion, with the result being the rules effectively form a decision tree. The usual inference method in these shells is backward chaining. Examples of rule network shells include CRYSTAL or VP-EXPERT.

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. Examples of commercially successful pattern matching shells include Xi Plus and Savoir.

It should be noted that at the time of writing, many shells are no longer sold or supported commercially; the surviving shell vendor companies have generally moved towards offering "web-based technology solutions" or "information management" solutions. Some likely sources of shells are given at the end of this article.

A KBS application developed using a shell

A good example of a KBS application developed using a shell is "Latent Damage Law - the Expert System", developed using CRYSTAL by Mason's solicitors. This system guides users through the complexities of UK legislation covering claims for damages that are discovered outside the period of normal contract law (for example, faults in buildings that do not manifest for fifteen or twenty years). CRYSTAL was a suitable tool for this application because legislation can be represented adequately as a set of production rules, assuming that expert opinion is available to assign values to terms such as "reasonable time" and "sufficient notice". This system can be obtained by purchasing a book describing the system's development; a disc is included with the book.

-
-
-

Bibliography

G. Anderson, A. Casson, A. Macintosh, R. Rae, B. Gleeson, and S. Carter (1996). FORMATION: Knowledge-Based Layout of Classified Telephone Directories. In *Applications and Innovations in Expert Systems IV, Proceedings ofBCS Expert Systems '96,* Cambridge, UK. [More details on the FORMATION system]

J. L. Alty. Expert system building tools (1989). *Topics in Expert System Design.* [An early survey of KBS tools]

P.W.H. Chung and J. Kingston (1987). State of the art knowledge based system toolkits. Technical Report AIAI-TR-54, AIAI. www.aiai.ed.ac.uk [More details on top-range toolkits]

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

M. Davies and K. Owen (1990). *Personnel Selection Screening: Graduates for Management.* HMSO UK. Part of the Department of Trade and Industry's Expert System Opportunities compilation. [Description of the GASS system]

E. Friedman-Hill (1999). Jess: The Java Expert System Shell, herzberg.ca.sandia.gov/jess/. [Description, manual, and downloadable files for JESS]

E. Feigenbaum, P. McCorduck, and H.P. Nii (1988). *The Rise of the Expert Company*. Macmillan. [A readable review of KBS from a management viewpoint. Includes details of the Authorizer's Assistant project]

C. Forgy (1982). Rete: A fast algorithm for the many-pattern/many-object pattern-match problem. *Artificial Intelligence,* 19(1). [The definitive description of the RETE algorithm]

C. Hall (1997-98). Intelligent Internet Systems: Parts 1 and 2. *Intelligent Software Strategies* [Surveys of recently released Internet-capable KBS tools]

R. Inder (1987). The State of the ART. Technical Report AIAI-TR-41, AIAI. [Description of Inference ART]

R. Inder (1999). CAPE: Extending CLIPS for the Internet. In *Research and Development in Intelligent Systems XVI: Proceedings of ES99, the 19th SGES International Conference on Knowledge Based Systems and Applied Artificial Intelligence,* Peterhouse College, Cambridge. Springer-Verlag. [Description of CAPE]

L. Johnson and E. Keravnou (1985). *Expert Systems Technology: A Guide.* Abacus Press, Cambridge, Mass. 02139. [General reference to early KBS applications, including MYCIN, INTERNIST and CASNET]

P. Klahr, J. Bak'in, F. Dashiell, J. Dzierzanowski, B. Gokhman, G. Hudkins, L. Koff, J. Mela, C. Nishiyama, L. Piketty, B. Ramesh, and L. Miller (1987). The Authorizer's Assistant: A Large Financial Expert System Application. In *Proceedings of the Third Australian Conference on Applications of Expert Systems,* pages 11-32, Sydney, Australia. Invited Keynote Paper. [Description of the Authorizer's Assistant from a technical/academic viewpoint]

P. J. Kline and S. B. Dolins (1989). *Designing expert systems : a guide to selecting implementation techniques.* Wiley. [A book that contains many "rules of thumb" to help in designing KBS and selecting KBS tools]

C MacNee (1992). PDQ: A knowledge-based system to help knowledge-based system designers to select knowledge representation and inference techniques. Master's thesis, Dept of Artificial Intelligence, University of Edinburgh. [Describes the design and development of the PDQ system]

C. J. Price (1990). *Knowledge Engineering Toolkits.* Ellis Horwood. [An early survey of available KBS toolkits]

R. Power, S. Reynolds, J.K.C. Kingston, I. Harrison, A. Macintosh, and J. Tonberg (1997). Expert Provisioner: A Range Management Aid. In *Applications and Innovations in Expert Systems V, Proceedings of BCS Expert Systems* '97, Cambridge, UK. [Details of the Expert Provisioner system]

P.Capper, R.E. Susskind, and Lord Justice Neill (1988). *Latent Damage Law - the Expert System.* Butterworths Law. ISBN: 040602362X. [Details of the Latent Damage Law system. The system itself is included with the book.]

S. Robertson and J. Kingston (1997). Selecting a KBS Tool using a Knowledge Based System. In *Proceedings of the Joint 1997 Pacific Asian Conference on Expert Systems / Singapore International Conference on Intelligent Systems (PACES/SPICIS '97)*, Singapore. [Description of a project to produce a KBS to help with KBS tool selection]

J. Rothenberg (1989). Expert system tool evaluation. *Topics in Expert System Design.* [A survey of KBS tools]

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

Biographical Sketch

John Kingston is a Senior Informatics Research Fellow in the Artificial Intelligence Applications Institute (AIAI), which is a part of Centre for Information Systems and their Applications, an institute within the Division of Informatics at the University of Edinburgh. After graduating from the University of Durham with a B.Sc (Hons) in Psychology, he completed a Master of Science in Knowledge Based Systems at the University of Edinburgh in 1986.

In his 15 years at AIAI, John has been involved in AI work with various companies and organizations, ranging from the UK Health and Safety Executive to the US Air Force and from Unilever to the University of Edinburgh. The key to the success of John's work has been his use and application of the CommonKADS methodology for knowledge analysis and KBS design. Recently, John's interests have moved to applying a generalised version of the CommonKADS approach (known as *multi-perspective modelling*) to knowledge management, via the capture and effective representation of corporate knowledge assets. John's research interests include techniques for capturing knowledge and modelling knowledge, methods for distributing knowledge (particularly intelligent Internet-based software), and the development of real-world applications which verify and exemplify all the aforementioned techniques.