AI Magazine Volume 8 Number 4 (1987) (© AAAI)

An Assessment of Tools for Building Large Knowledge-Based Systems

William Mettrey

A number of tools that support the development, execution, and maintenance of knowledge-based systems are marketed commercially Many of these tools, however, are designed for applications that can be executed on personal computers and are not suitable for building large knowledge-based systems The market for knowledge engineering tools designed for applications that require the computational power of a Lisp machine or an engineering workstation is dominated by a few vendors This article is an assessment of the current state of tools used to build large knowledge-based systems This assessment is based on the collective strengths and weaknesses of several tools that have been evaluated In addition, an estimate is made of the features that will be required in the next generation of tools

Knowledge engineering tools are software systems that support the development, execution, and maintenance of knowledge-based systems (KBSs). The market for tools used to build large KBSs is currently dominated by a few vendors with offerings that support multiple knowledge representation and inferencing paradigms, the ability to pursue hypothetical reasoning, truth maintenance, object-oriented programming, extensive debugging aids, and graphic interfaces (Richer 1986).

Current commercial tools can be thought of as the third generation of knowledge engineering environments (Hayes-Roth 1984). The research community produced first-generation KBSs by developing Lisp programs that intermixed knowledge and control structures. Second-generation KBSs were also developed in research environments. The designers of second-generation KBSs recognized the value of separating the knowledge base from the inferencing process, thus extending the utility of a shell from a specific application to a class of related problems. The current generation of commercial tools, for the most part, are extensions of these research efforts.

As a result of an in-depth study (Knowledge Systems Corp. 1986) of commercially supported tools for large KBSs, a number of conclusions were formed. It is not the intent of this article to compare the relative strengths and weaknesses of the tools that were evaluated but rather to assess the current state of commercial knowledge engineering tools and estimate features which will be required in the next generation. The categories that are addressed include knowledge representation, inferencing and control, development environments, and delivery environments. Although features from a number of tools are considered, most examples are drawn from five of the most frequently used tools for building large systems: (1) Automated Reasoning Tool (ART) marketed by Inference Corporation, (2) Knowledge Engineering Environment (KEE) marketed by IntelliCorp, (3) KnowledgeCraft marketed by Carnegie Group Inc., (4) S 1 marketed by Teknowledge Inc., and (5) VAX OPS5 marketed by Digital Equipment.

Knowledge Representation

At a minimum, commercial knowledge engineering tools support the use of rules for knowledge representation. In addition, tools designed for large KBSs also provide other knowledge representation approaches, such as frames, objects, and the ability to extend the knowledge base to support hypothetical reasoning.

Rules

Rules provide a modular and uniform approach to knowledge representation. Tools that support rules as their sole representation paradigm are relatively simple to learn and use. As rule-based systems grow, however, they become increasingly difficult to understand and maintain (van de Brug, Bachant, and McDermott 1986). In addition, tools that offer only a rule-based approach are less effective in problem domains that exhibit complex relationships among a number of objects, activities, and events (Brown 1985). The intermixing of rules that describe objects and relationships with rules which represent heuristic knowledge increases the complexity of a knowledge base

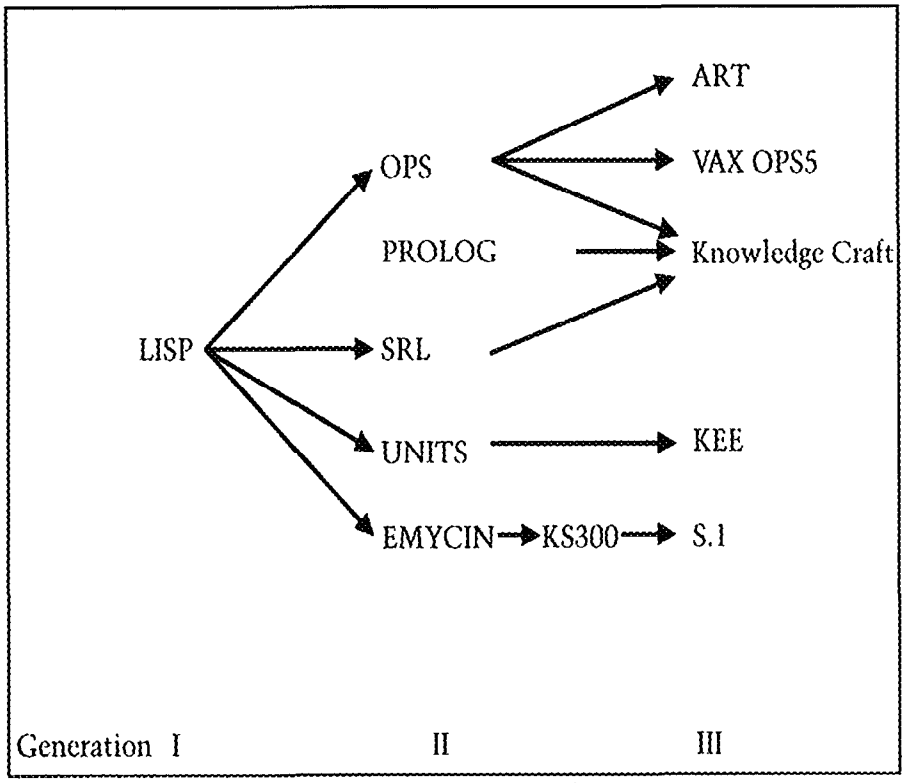

Figure 1 Evolution of Knowledge Engineering Tools

Frames

Several tools allow multiple representational schemes. These tools support a version of frames (Fikes and Kehler 1985) in addition to rules. S.1 (Teknowledge 1985b) uses frames as templates for classes or logical groupings of entities. When a class instance is created, S.1 generates a frame that inherits the attributes (slots) of the class but does not inherit any values. ART (Inference Corporation 1987), KEE (IntelliCorp 1985), and KnowledgeCraft (Carnegie Group 1987) support inheritance of slots and values as well as the ability to attach metaknowledge to slots. Metaknowledge can be used to constrain values, govern the actions of procedures attached to slots, and redefine inheritance behavior.

Objects

ART, KEE, and KnowledgeCraft also support objects as a knowledge representation approach. Each object is represented by a frame with methods (procedures) attached to the frame's slots. Objects communicate with one another by sending messages. When an object receives a message, the message is interpreted, and the appropriate method is activated. Objects and message passing provide a natural way to represent entities in problems involving complex relationships (Stefik and Bobrow 1986). The use of object-oriented programming in the environments provided by ART, KEE, and KnowledgeCraft requires proficiency in Lisp in order to develop the required methods.

Hypothetical Reasoning

ART's viewpoints, KnowledgeCraft's contexts, and KEE's KEEworlds are mechanisms that support hypothetical reasoning. In effect (though not in fact), multiple knowledge bases are created and maintained during a consultation (Clayton 1985). When a situation suggests several interpretations or outcomes, virtual copies of the knowledge base are generated to pursue subsequent reasoning about the possibilities. Each copy can be thought of as a separate model of the domain of interest. This approach is useful for representing hypothetical situations and situations that vary over time.

Requirements for Knowledge Representation

Knowledge representation paradigms supported by commercial tools are adequate for many of the problems that are being attempted. However, a number of unresolved issues exist. Among these issues are knowledge-acquisition needs, representation of causal models, representation of spatial and temporal relations, a requirement for improved methods of handling incomplete and uncertain knowledge, a requirement for tools that understand and enhance their knowledge base, and a requirement for methodologies to evaluate knowledge representation issues.

The effectiveness of a KBS is dependent to a large extent on the amount of domain-specific expertise it embodies. Much of this expertise is symbolic and heuristic and is usually not well formalized It is the knowledge engineer's task to capture this expertise and codify it in a form that is usable by a knowledge engineering tool. Knowledge acquisition is currently one of the major bottlenecks in the building of KBSs (Hoffman 1987). Although research systems have been built that aid knowledge acquisition (Davis 1976; Eshelman and McDermott 1986), similar facilities are nonexistent in commercial tools.

Causal models specify relations among actions and events. They use first principles to provide a detailed specification of how a complex device or process functions. EL (Stallman and Sussman 1979) is an example of a causal KBS developed as a research prototype. Given a description of a circuit schematic, EL analyzes the circuit and determines the values of voltage and current at specified points. EL's expertise includes general principles of electronics (Ohm's law, Kirchoff's law) and circuit component characteristics (the functional properties of transistors). Although research projects, such as EL, exist in which causal representations are central, knowledge of this type is generally difficult to represent and reason about using commercial tools.

Spatial models are characterized by the need to represent and reason about the size, shape, and position of

objects; a number of design problems fall into this category. XCON (McDermott 1982), a KBS for configuring VAX computers from customer orders, decides which components must be added to produce a complete operational system and determines spatial relationships among these components. XCON generates diagrams of the spatial relationships for use by technicians in assembling the VAXen.

Temporal models involve reasoning about problem domains that vary over time. Planning, prediction, and problems that deal with continuous processes or continuous flows of data are examples of domains which require temporal models. Representations required by temporal models differ from those required for static analysis (Fagan 1980). Temporal models exhibit the need for multiple contexts in order to address the problem of representing different time intervals. As is the case with causal models, the development of KBSs for domains that require spatial and temporal relations is difficult to accomplish with current knowledge engineering tools.

Tools that support the specification of uncertain knowledge tend to be designed for specific classes of problems. Although a few tools utilize fuzzy logic or Bayesian approaches, the majority of tools that support the specification of uncertain knowledge use certainty factors The calculation of certainty factors is currently an ad hoc process that is usually modeled after the work in MYCIN (Buchanan and Shortliffe 1984). A need exists for further research of the representation and manipulation of uncertain knowledge.

The effectiveness of a KBS can be improved by the knowledge engineer's addition and refinement of knowledge Future generations of tools should have the ability to assist in the modification and extension of their knowledge base These tools should "learn" as they execute, generating new rules and objects and modifying old ones This type of system might make decisions, such as creating several new rules to restrict an overly general rule; create one rule to replace several that have been applied in a sequence; and formulate metarules to guide its own behavior. The feasibility of systems improving their performance using metarules and self-modification has been demonstrated in research projects (Green and Westfold 1982) but has not progressed to the point of being supported by commercial tools.

Table 1. Knowledge Representation.

	ART	KEE	KnowledgeCraft	S.1	VAX OPS5
Rules	yes	yes	yes	yes	yes
Frames	yes	yes	yes	limited	no
Inheritance	yes	yes	yes	no	no
Uncertainty	no	no	no	yes	no
Hypothetical Reasoning	yes	yes	yes	no	no
Meta-knowledge	yes	yes	yes	no	no
Knowledge Acquisition Aids	no	no	no	no	no
Self-learning Knowledge Base	no	no	no	no	no

The question of multiple knowledge representation schemes versus a single approach presents a trade-off between the naturalness of representation and the ease of learning and using a tool Representational needs are dictated by the nature of the problems that are being addressed. Selecting tools that support multiple representational schemes for an application adequately described by a few hundred rules might be unwarranted. Applications that are characterized by complex dependencies among objects such as spatial and temporal relationships, however, are difficult to represent with a tool which only supports rules. Methodologies are needed to match representational paradigms to problem types and evaluate the efficiency of these representations. Given the current state of knowledge representation and the continuing debate over the constitution of semantic nets, frames, and other representational structures (Brachman and Levesque 1985), these methodologies appear to be a long-term objective.

Inferencing and Control

The suitability of a knowledge engineering tool for various classes of problems is influenced by the inferencing paradigms it provides. Tools intended for building large KBSs provide multiple paradigms, including rule-based inferencing, procedural control, object-oriented programming, access-oriented programming, truth maintenance, and knowledge-based simulation

Rule-Based Inferencing

Among the rule-based inferencing features that characterize tools designed for large KBSs are support for both forward and backward chaining, sophisticated conflict-resolution strategies, and the ability to compile rules.

Forward chaining and backward chaining are supported by commercial tools in varied ways ART, KnowledgeCraft, and VAX OPS5 adhere to the pure production system model in their support of forward chaining This approach makes a rule eligible to fire whenever its conditions are matched by elements in a global store called, *working memory* KEE implements forward chaining by making rules eligible to fire that have conditions which match asserted facts in any of the knowledge base's frames or in an unstructured facts list. ART implements backward chaining by allowing a single backward-chaining goal pattern as one of the conditions in a forward-chaining rule. S.1's support of backward chaining offers a high level of abstraction, insulating the user from details such as backtracking By contrast, KnowledgeCraft implements backward chaining by supporting a version of Prolog This approach allows the knowledge engineer to control backtracking by using features such as "cut" and "fail."

ART, KEE, and KnowledgeCraft support a mixed-strategy approach by allowing both forward and backward chaining to be applied to the same problem. The transfer of control between the two strategies is accomplished implicitly or explicitly depending on the tool. Forward chaining is ART's default inferencing strategy and is executed until a backward-chaining goal is encountered. The choice of forward or backward chaining in KEE and KnowledgeCraft is specified by user function calls.

Current tools also vary in the manner in which conflict resolution (McDermott and Forgy 1978) is supported. Conflict resolution is required when several rules are in the conflict set (that is, are candidates to fire), and some strategy must be used to select among them KnowledgeCraft's CRL-OPS and Digital's VAX OPS5 implement conflict resolution by first attempting to select the rule that has been instantiated by the most recent data in working memory. If recency does not resolve the conflict, these tools select a rule instantiation based on the number and complexity of conditions that were matched. KEE allows the user to select a conflict-resolution strategy from a number of options, including least premise complexity, greatest premise complexity, least weight, or greatest weight ART selects the instantiated rule with the highest user-specified priority

Support for rule compilation is a feature that distinguishes knowledge engineering tools intended for building large KBSs from tools oriented toward smaller problems. The term compile in this sense refers to a mapping from a rule's external form to an internal representation. This internal representation is designed to reduce the number of pattern-matching operations required to place rules in the conflict set. An obvious approach to building a conflict set is to match each rule in the knowledge base against all the working memory elements on each recognize-act cycle. This approach is prohibitive for a large rule base (Forgy 1979).

ART, KnowledgeCraft's CRL-OPS, and OPS5 use variations of the Rete matching algorithm in conjunction with rule compilation (Forgy and Shepard 1987). The Rete algorithm exploits two properties common to production systems: The contents of working memory change very little on each iteration of the recognize-act cycle, and rules contain a number of similar conditions and terms that are to be matched against working memory. The algorithm is implemented by compiling rule conditions into a pattern-matching network that enables the matching process to factor and share similar tests Timing tests conducted by the National Aeronautics and Space Administration (Riley 1986) exhibited significant advantages for forward-chaining tools that use rule compilation versus those tools which did not support rule compilation.

When compared to the current state of software engineering, the immaturity of knowledge engineering is apparent.

Procedural Control

Several tools furnish language constructs that enable a knowledge engineer to specify control strategies. S.1 supports a Pascal-like procedural language that can be used to create objects and assign values to their attributes, query users and display results, and initiate backward chaining. ART provides constructs such as "for" and "while" that can be specified in rules to effect iteration. In addition, most tools allow the user to escape to external procedures, either to achieve functionality not supported by the tool or to interface with other systems.

Object-Oriented Programming

ART, KEE, and KnowledgeCraft support object-oriented programming The approaches of these tools are similar in that objects are represented as frames with methods attached to specified slots. Methods are inherited in the same manner as conventional slot values. Inheritance allows methods to be shared by all objects that belong to a particular class. In many cases, methods must be developed by the knowledge engineer to meet specific needs. Method libraries provided by current tools are not as extensive as those supplied by systems such as Smalltalk (Goldberg and Robson 1983) and Objective-C (Cox 1986).

Access-Oriented Programming

The basic primitives of access-oriented programming are procedures (demons, active values) that are activated as side effects when specified variables are accessed (Bobrow and Stefik 1986). Methods and demons are conceptually very similar. The chief distinction between them is the way they are activated Methods are activated by a message, and demons are activated when their associated values are accessed. ART, KEE, and KnowledgeCraft support demons. Demons are particularly useful when the knowledge engineer wants to make changes at dynamically determined points in an execution.

Truth Maintenance

Truth maintenance (Doyle 1982) is a means of keeping track of beliefs and their justifications developed during an inferencing process. If contradictions occur, the lines of reasoning associated with the contradictions and all conclusions resulting from them can be retracted Both ART and KEE support truth maintenance systems. ART allows the knowledge engineer to specify contradiction rules that are automatically applied whenever the specified inconsistency exists. KEE's truth maintenance system is integrated with its KEEworlds facility. By keeping track of the justifications that lead to a belief being accepted, KEE is able to determine which KEEworlds are consistent with the belief.

Knowledge-Based Simulation

Knowledge-based simulation combines knowledge representation and inferencing with discrete-event simulation (Oren 1985). Several requirements have motivated this approach From the simulation perspective, many complex problems lend themselves to symbolic and object-oriented approaches. These approaches allow the construction of models using symbolic programming, object-oriented programming, and frame-based knowledge representation with inheritance (Klahr 1985; Middleton and Zanconato 1985). From the KBS perspective, a simulation component, combined with the various inferencing paradigms, increases the power of the reasoning process. Planning problems in which temporal reasoning is organized in an event-driven framework

lend themselves to this approach (Erickson 1985).

Both SIMKIT (Stelzner, Dynis, and Cummins 1987) and KnowledgeCraft provide support for knowledge-based simulation. SIMKIT, which was built using KEE, supports event creation and scheduling, generation of random variables, the manipulation of graphic icons, and facilities for collecting and reporting the results from a simulation. KnowledgeCraft supports knowledge-based simulation by providing a multiqueue event manager that schedules events to occur using a simulated clock. This capability is integrated with CRL, KnowledgeCraft's knowledge representation language.

Requirements for Inferencing and Control

As is the case with knowledge representation, inferencing paradigms supported by current tools are adequate for many of today's problems. A number of needs currently exist in this area, however, including improved integration of inferencing techniques in hybrid tools, inferencing approaches which provide several levels of abstraction, and methodologies that determine the efficiency of inferencing approaches and furnish guidance about which combination of paradigms should be applied to specific problems.

Tools that support several inferencing paradigms find it difficult to present a user interface that is as easy to understand and use as tools whose design is based upon a single approach. Some hybrid tools support forward chaining and backward chaining as independent subsystems that are invoked by user calls, much like a run-time library in conventional programming systems. This approach sometimes results in two independent inferencing strategies that are not sensitive to one another and leave to the user the task of integrating them into the same problem solution. Another approach is to treat forward chaining as the primary strategy and initiate backward chaining at certain points in the inferencing process to assist forward chaining. Although this approach achieves some degree of integration, current implementations give the appearance of being developed by grafting backward chaining upon an existing forward-chaining system.

Table 2. Inference and Control.

	ART	KEE	KnowledgeCraft	S.1	VAX OPS5
Forward Chaining	yes	yes	yes	no	yes
Backward Chaining	yes	yes	yes	yes	no
Mixed Strategy	yes	yes	yes	no	no
Rate Network	yes	no	yes	no	yes
Demons	yes	yes	yes	no	no
Truth Maintenance	yes	yes	no	no	no
Message Passing	yes	yes	yes	no	no
Meta-control	no	no	no	no	no

Tools that furnish several levels of abstraction for inferencing are also needed. At a high level of abstraction, features such as menus, online help, online tutorials, and guidance about which techniques to use based on problem characteristics are required. These features allow domain experts who lack experience with knowledge engineering tools to develop and maintain models. A lower level of abstraction for experienced knowledge engineers should also be provided This level would allow the specification of metacontrol information to guide inferencing at various points during processing.

Methodologies are required for evaluating the efficiency of inferencing approaches and matching them to problem types. In particular, metrics are needed which provide information to be used in assigning multiple inferencing paradigms to specific problems and which guide in partitioning the problem so that the various strategies can be applied in an effective manner.

Development Environments

A *development environment* is defined here as a set of facilities that enable a knowledge engineer to create, test, and validate a KBS. Development environments are addressed in terms of knowledge base management and debugging.

Knowledge Base Management

KBSs are usually built incrementally. Utilities are required that support the addition of new knowledge and the removal of invalid or redundant knowledge in order to effectively build and maintain these systems. ART, KEE, and KnowledgeCraft provide support for windows and mouse-selectable menus to aid in the development and maintenance of knowledge bases. These tools allow the user to create and shape windows and specify their positioning. Windows are used to enter information into the knowledge base and display knowledge base entities. In addition, the execution of knowledge base utilities from menus is supported. Although a number of tools provide access to an editor from within their environment, the functionality of the editor is usually dependent on the system that hosts the tool.

Debugging

Debugging is an area where some commercial knowledge engineering tools exhibit a superiority over traditional software development environments. This advantage is in part the result of the Lisp environments in which many of the tools are implemented. Lisp programmers have traditionally required extensive development and debugging facilities because of the exploratory nature of the applications that have been addressed with the language. Some tools support facilities such as execution traces, the ability to halt an execution to examine and modify values and then resume, the capability to back up and reexecute during testing, the ability to save and retrieve test cases as well as built-in explanation facilities that answer questions concerning how and why specific conclusions are reached. ART and KEE provide graphic tracing of rule activity during inferencing. KEE and KnowledgeCraft support a graphic display of frames in the form of an inheritance network. Frames can be added to, and deleted from, the knowledge base by manipulating the displays with a mouse. Graphic displays can also be used to edit the slots and values of frames. In addition to

Table 3. Development Environment.

	ART	KEE	KnowledgeCraft	S.1	VAX OPS5
Internal Editor	yes	yes	yes	yes	no
Help Menus	yes	yes	yes	yes	no
Saved Cases	no	no	no	yes	no
Trace and Probes	yes	yes	yes	yes	yes
Breakpoints	yes	yes	yes	yes	yes
Performance Measurement	yes	no	no	no	yes
Graphics Interface	yes	yes	yes	yes	no
Graphics Toolkit	yes	yes	yes	no	no
Cross Referencing	no	no	no	no	no
Version Control	no	no	no	no	no

the interfaces provided for development, ART, KEE, and KnowledgeCraft furnish graphic tool kits that allow the knowledge engineer to develop custom interfaces for end users.

Requirements for Development Environments

As a knowledge base grows, it becomes difficult to determine manually whether new knowledge is redundant or conflicts with existing knowledge. Utilities that "understand" the semantics of the knowledge base and perform consistency checking are needed. Features of this type currently exist in only a few systems. TEIRESIAS (Davis 1976) is able to analyze rules, make suggestions regarding their completeness and consistency, and help the knowledge engineer debug them using its own rule heuristics. CHECK (Nguyen et al. 1987) performs static analysis of a knowledge base and is able to detect redundant, conflicting, and circular rules. Versions of CHECK exist that diagnose knowledge bases for LES (Laffey, Perkins, and Nguyen 1986) and ART.

Cross-reference utilities that provide information about relationships in a knowledge base are needed. Cross-reference information is valuable when there is a need to retrieve all components that contain or reference a given entity or when it is necessary to determine which entities in the knowledge base are affected when a given instance is added, modified, or deleted.

Utilities are also needed that maintain different versions of the knowledge base and record the updates as they are made. EMYCIN provides this type of support. When a rule is changed, the EMYCIN editor automatically stores the modification date and user's name with the rule for later reference. This feature is particularly useful when several people are involved in the maintenance of a large knowledge base.

ART and VAX OPS5 are among the few tools that aid the knowledge engineer in monitoring and evaluating the performance of KBSs. ART provides statistics concerning the number of inferences occurring during a run, pattern memory usage, and rule activations and firings. VAX OPS5 reports the number of times a rule fires, the amount of central processing unit (CPU) time used to execute the left-hand side of a rule, and the amount of CPU time used to execute the right-hand side of a rule. The identification of entities that consume large amounts of resources is required in order to apply KBSs to time-critical applications. As KBSs move from stand-alone environments and become components of information systems, performance issues become important. Performance monitoring and evaluation facilities can also be used to determine the relative efficiency of the various inferencing methods and representation approaches. An allied need exists for standard test case suites that can be used to evaluate the relative performance of tools.

For knowledge engineering to progress beyond an art and truly deserve the engineering label, a set of principles and methodologies must be formulated and adopted. When compared to the current state of software engineering, the immaturity of knowledge engineering is apparent. The relative youth of knowledge engineering has not yet allowed a sufficient base of experience to be collected, cataloged, and extrapolated to sound methodologies. The classic software engineering approach consists of developing a system using a specification of the problem in conjunction with top-down techniques. In contrast, most KBSs are developed incrementally in a bottom-up approach for applications that are not well understood. Often, the building of a prototype system serves as the specification for the problem. Despite differences in the two disciplines, it appears that some software engineering principles are applicable to KBSs (Zualkernan, Tsai, and Volovik 1986). In addition, new methodologies that address the unique features of knowledge engineering (for example, knowledge acquisition) must be developed. A necessary step toward achieving these goals is collecting and cataloging user experience with the various knowledge engineering tools.

Delivery Environments

A *delivery environment* is defined here as the hardware and software that enable an end user to execute a KBS. A problem frequently encountered by developers of large KBSs is the selection of cost-effective hardware for the delivery environment. Candidate hardware must be supported by the appropriate knowledge engineering tool and provide sufficient system resources to execute the KBS at the required level of response; its cost must allow a meaningful return from the application. Potential delivery systems for large KBSs include Lisp machines, mainframes, workstations, and personal computers.

Most of the tools for building large KBSs were originally developed for Lisp machines. These machines are designed to give single users access to all the system's resources. Constraints that tend to inhibit development, such as restricting access to the file system, have been avoided. However, the same features that aid rapid prototyping in a development environment can result in a lack of security in a delivery environment. Also, the single-user orientation of Lisp machines becomes a cost consideration in delivery environments when the KBS is distributed to a number of users. KBSs currently delivered on Lisp machines tend to be highly leveraged applications that are distributed to a small number of users.

KBSs are beginning to appear on hardware that has traditionally hosted data processing applications. Tools such as Aion (Aion Corporation 1986),

Expert System Environment (International Business Machines 1986) and S.1 support the delivery of KBSs on IBM mainframes. These tools, written in languages other than Lisp, are designed for applications that are integrated with conventional software systems. The ability to interface with the host operating system, access large databases, and be called by other applications becomes increasingly important as KBSs progress beyond stand-alone applications and form components of large software systems.

Many of the tools developed for Lisp machines have been ported to workstations such as the microVAX, SUN, and Apollo systems. Some of these tools have remained Lisp based. Other tools such as S.1 and ART have been rewritten in C in an attempt to achieve increased run-time performance and easier integration with existing applications.

A number of tools support development and delivery of small and medium KBSs on personal computers (Teknowledge 1985a; Texas Instruments 1985; Henson 1987). In addition, IntelliCorp provides PC-HOST, a delivery option that hosts a run-time version of KEE on a VAX and executes the user interface on a personal computer. Several vendors are planning delivery options that take advantage of increases in the computational speed and memory capacity of 32-bit personal computers. IntelliCorp *IntelliNews* (1987) plans a run-time version of KEE that will execute on Intel 80386-based personal computers.

Although a number of delivery options currently exist, tool vendors continue to search for better solutions. Factors such as hardware price-performance improvements, very large scale integration Lisp chips, improved Lisp compilers, and tools implemented in conventional languages will influence future delivery environments. At present, no single direction clearly dominates.

Requirements for Delivery Environments

Features required of a tool in a delivery environment differ from those needed in a development environment. Development environments require facilities that enhance editing and debugging as well as the ability to experiment with different knowledge representation and inferencing approaches. Delivery environments require good end-user interfaces, efficient execution, and the ability to interface with other systems. A number of tools (for example, ART, KEE, and KnowledgeCraft) provide facilities for building end-user interfaces. The areas where delivery tools must show improvement in order to effectively use KBSs for large applications include improved execution speed, better integration with traditional information-processing systems, improved input-output, and access to operating-system services.

Table 4. Delivery Systems.

Lisp Machines	ART	KEE	KnowledgeCraft	S.1	VAX OPS5
Symbolics	LISP	LISP	LISP	LISP	—
LMI	LISP	LISP	—	—	—
TI Explorer	LISP	LISP	LISP	—	—
XEROX	—	LISP	—	LISP	—
Workstations					
VAX	C	LISP	LISP	C	BLISS
APOLLO	—	LISP	LISP	C	—
SUN-3	LISP	LISP	LISP	C	—
IBM PC/RT	—	LISP	—	C	—
NCA Tower	—	LISP	—	C	—

Improvements in execution speed are needed to effectively address a number of applications. The feasibility of using a KBS consisting of several thousand rules in a critical real-time application is probably beyond the current state of the art. Brownston et al. (1985) estimate that most production systems run from one to two orders of magnitude slower than procedural programs. Goals for the Department of Defense-sponsored Strategic Computing Initiative (Hayes-Roth 1985) include increasing the size of KBSs to 10,000 or more rules and increasing solution speeds by two orders of magnitude or more. Some of these required efficiency gains might result from the use of parallel processing and language-directed architectural approaches (Forgy and Gupta 1986, Blelloch 1986). Production systems in particular appear to lend themselves to parallel execution. Operations such as pattern matching, rule selection, conflict resolution, and rule firings can theoretically be carried out in parallel. To achieve parallelism, extensive analysis is required to avoid generating invalid information in the knowledge base.

Current KBSs typically operate in a stand-alone consultation mode. As the technology matures, there is a need to support features that allow KBSs to be incorporated into traditional software systems and interface with applications such as database management systems. A factor that inhibits these objectives is the lack of adequate execution-time input-output facilities furnished by tools. Direct support of input-output facilities by current tools is primitive by most standards. It consists almost entirely of being able to accept user responses from a terminal and display results to a terminal. In order to perform basic input-output operations (for example, reading or writing a sequential file), the knowledge engineer must often use Lisp functions or resort to external procedures.

In order for KBSs to be used effectively in applications on multiuser computing systems, tools must allow access to a number of operating-system services. In particular, as KBSs find increased use in real-time applications, they will need the ability both to respond to interrupts and to be scheduled in multitasking environments that allow a number of cooperating processes to synchronize and share data.

Summary

Most knowledge engineering tools that currently dominate the industrial market are the result of research efforts which were later commercial-

. . . a number of unresolved issues exist. . . . knowledge-acquisition needs, representation of causal models, representation of spatial and temporal relations, a requirement for improved methods of handling incomplete and uncertain knowledge, a requirement for tools that understand and enhance their knowledge base, and a requirement for methodologies to evaluate knowledge representation issues.

ized. These tools have evolved as products to the point where they collectively support an impressive set of knowledge engineering features. These features include multiple knowledge representation and inferencing approaches, the ability to pursue hypothetical reasoning, object-oriented programming, support of truth maintenance, extensive debugging aids, and graphic interfaces. Although a number of significant KBSs have been built using these tools, they have required the efforts of skilled knowledge engineers. The goal of having a domain expert with minimal computer expertise create and maintain a KBS is not feasible with this generation of tools. To reach this goal, a number of extensions and improvements are required.

First is improved facilities for acquiring and testing knowledge. These facilities must simplify the task of transferring knowledge from the domain expert to the tool's knowledge base and help maintain the integrity of the knowledge base by supporting semantic checking, cross-referencing, and version control.

The next improvement is knowledge representation facilities that support reasoning from first principles, allow natural representation of spatial and temporal relations, and furnish improved methods for specifying uncertain knowledge. Third is improved integration of inferencing approaches with the flexibility to allow an inexperienced user to control inferencing at a high level of abstraction and a knowledge engineer to specify low-level metacontrol information.

Fourth, methodologies are needed to evaluate knowledge representation and inferencing issues and contribute to the development process in a manner comparable to current software engineering practices. Then there are improved execution speeds that allow the use of KBSs in critical real-time applications and performance monitoring facilities which enable the tuning of a model's execution. Finally, features are needed that allow the seamless integration of KBSs with existing applications, such as database management systems and conventional language applications. These features must include improved input-output facilities and access to operating-system services.

It will not be possible to achieve all these goals in the framework provided by current tools. Robust and well-integrated implementations of these features will require new designs. These advances will accelerate the proliferation of KBSs both as stand-alone systems and as components in traditional software systems.

Acknowledgments

I wish to thank Digital Equipment Corporation for funding the tool study and Knowledge Systems Corporation for sponsoring my involvement In addition, I would like to thank the referee for numerous constructive comments.

References

Aion Corporation 1986 Aion Development System Reference Manual, Aion Corporation

Blelloch, G E 1986 CIS: A Massively Concurrent Rule-Based System In Proceedings of the Fifth National Conference on Artificial Intelligence, pp 735-741. Los Altos, Calif.: Morgan-Kaufmann

Bobrow, D. G , and Stefik, M. J. 1986. Perspectives on Artificial Intelligence Programming *Science* 231:951-957

Brachman, R J , and Levesque, H J 1985. *Readings in Knowledge Representation* Los Altos, Calif : Morgan-Kaufmann.

Brown, J S 1985. The Low Road, the Middle Road, and the High Road In *The AI Business*, eds P H. Winston and K A Prendergast, 81-90 Cambridge, Mass.: MIT Press.

Brownston, L ; Farrel, R ; Kant, E.; and Martin, N 1985. *Programming Expert Systems in OPS5* Reading, Mass.: Addison-Wesley

Buchanan, B., and Shortliffe, E 1984 *Rule-Based Expert Systems, The MYCIN Experiments of the Stanford Heuristic Programming Project* Reading, Mass : Addison-Wesley

Carnegie Group 1987. KnowledgeCraft CRL Technical Manual, Version 3 1, Carnegie Group

Clayton, B D 1985 ART Programming Tutorial, Volume Two: A First Look at Viewpoints. Los Angeles, Calif.: Inference Corporation.

Cox, B. J. 1986. *Object-Oriented Programming: An Evolutionary Approach* Reading, Mass.: Addison-Wesley.

Davis, R 1976 Applications of Meta-Level Knowledge to the Construction, Maintenance, and Use of Large Knowledge Bases, Technical Report, STAN-CS-76-552, Dept of Computer Science, Stanford Univ

Doyle, J 1982 A Glimpse of Truth-Maintenance. In *Artificial Intelligence An MIT Perspective*, eds. P H. Winston and R H Brown, 119-135 Cambridge, Mass.: MIT Press

Erickson, S. A 1985 Fusing AI and Simulation in Military Modeling In *AI Applied to Simulation*, eds E J. H. Kerckhoffs, G. C. Vansteenkiste, and B. P Zeigler, 140-150 San Diego, Calif.: Society for Computer Simulation.

Eshelman, L , and McDermott, J 1986 MOLE: A Knowledge Acquisition Tool That Uses Its Head In Proceedings of the Fifth National Conference on Artificial Intelligence, 950-955. Los Altos, Calif : Morgan-Kaufmann

Fagan, L. 1980. VM: Representing Time-Dependent Relations in a Clinical Setting Ph D diss., Dept of Computer Science, Stanford Univ

Fikes, R , and Kehler, T 1985. The Role of Frame-Based Representation in Reasoning *Communications of the ACM* 28(9): 904-920

Forgy, C L , 1979. On the Efficient Implementation of Production Systems Ph.D. diss , Dept of Computer Science,

Carnegie-Mellon Univ.

Forgy, C. L., and Gupta, A 1986 Preliminary Architecture of the CMU Production System Machine. In Proceedings of Hawaii International Conference on System Sciences

Forgy, C. L., and Shepard, S J 1987 Rete: A Fast Match Algorithm *AI Expert* 2(4): 35-40

Goldberg, A , and Robson, D. 1983 *Smalltalk-80: The Language and Its Implementation* Reading, Mass : Addison-Wesley

Green, C , and Westfold, S J 1982 Knowledge-Based Programming Self-Applied In *Machine Intelligence 10: Knowledge-Based Systems,* eds J. E Hayes, Donald Michie, and Y-H Pao, 339-359 New York: Wiley

Hayes-Roth, F 1985 Rule-Based Systems. *Communications of the ACM* 28(9): 921-932.

Hayes-Roth, F 1984. The Knowledge-Based Expert System: A Tutorial. *IEEE Computer* 17(9):11-28

Henson, D. G 1987 Gold Works Expert System User's Guide Gold Hill Computers

Hoffman, R. H 1987. The Problem of Extracting the Knowledge of Experts from the Perspective of Experimental Psychology *AI Magazine* 8(2): 53-67.

International Business Machines 1986. ESCE/VM User Guide, SH20-9606, International Business Machines

Inference Corporation, 1987. ART Reference Manual, Version 3.0, Inference Corporation.

IntelliCorp, 1985. KEE Software Development System User's Manual, Version 3.0, IntelliCorp

IntelliNews 3(3) April 1987. Ed J. Dynis, IntelliCorp

Klahr, P. 1985 Expressibility in ROSS, An Object-Oriented Simulation System In *AI Applied to Simulation,* eds E. J H Kerckhoffs, G. C Vansteenkiste, and B P. Zeigler, 136-139. San Diego, Calif.: Society for Computer Simulation

Knowledge Systems Corporation 1986 Evaluation of AI Languages and Knowledge Engineering Environments: Phase II, Knowledge Systems Corporation.

Laffey, T J ; Perkins, W. A ; and Nguyen, T A 1986 Reasoning about Fault Diagnosis with LES. *IEEE Expert, Intelligent Systems and Their Applications* 1(1): 13-20

McDermott, J 1982. R1: A Rule-Based Configurer of Computer Systems. *Artificial Intelligence* 19: 39-88

McDermott, J., and Forgy, C 1978. Production System Conflict Resolution Strategies In *Pattern-Directed Inference Systems,* eds. D A Waterman and F Hayes-Roth, 177-199 New York: Academic

Middleton, S , and Zanconato, R. 1985 BLOBS: An Object-Oriented Language for Simulation and Reasoning. In *AI Applied to Simulation,* eds E J H Kerckhoffs, G. C Vansteenkiste, and B P Zeigler, 130-135 San Diego, Calif.: Society for Computer Simulation

Nguyen, T A.; Perkins, W A.; Laffey, T. J ; and Pecora, D 1987 Knowledge Base Verification *AI Magazine* 8(2): 69-75

Oren, T I. 1985 Artificial Intelligence and Simulation In *AI Applied to Simulation,* eds E. J H Kerckhoffs, G C Vansteenkiste, and B P Zeigler, 3-8 San Diego, Calif : Society for Computer Simulation.

Richer, M H. 1986. An Evaluation of Expert System Development Tools, Technical Report, KSL 85-19, Knowledge Systems Laboratory, Dept of Computer Science, Stanford Univ

Riley, G D 1986 Timing Tests of Expert System Building Tools, FM7 (86-51), Lyndon B Johnson Space Center, National Aeronautics and Space Administration

Stallman, R M., and Sussman, G J. 1979. Problem Solving about Electrical Circuits In *Artificial Intelligence: An MIT Perspective,* eds. P H. Winston and R H Brown, 33-91. Cambridge, Mass : MIT Press

Stefik, M , and Bobrow, D. 1986 Object-Oriented Programming: Themes and Variations. *AI Magazine* 6(4): 40-62

Stelzner, M ; Dynis, J ; and Cummins, F. 1987 The SIMKIT System: Knowledge-Based Simulation and Modeling Tools in KEE. IntelliCorp

Teknowledge 1985a. M 1 Reference Guide. Teknowledge.

Teknowledge. 1985b. S 1 Users Guide Teknowledge

Texas Instruments, 1985 Personal Consultant Expert System Development Tools, 2243763-0001, Texas Instruments

van de Brug, A ; Bachant, J ; and McDermott J 1986 The Taming of R1 *IEEE Expert* 1(3): 33-38.

Zualkernan, I ; Tsai, W T ; and Volovik, D 1986. Expert Systems and Software Engineering: Ready for Marriage? *IEEE Expert* 1(4): 24-30