KNOWLEDGE ENGINEERING: THE DESIGN OF INTELLIGENT COMPUTERS

Benjamin Wan-Sang Wah
Department of Electrical and Computer Engineering
and the Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1101 West Springfield Avenue
Urbana, IL 61801, U. S. A.
wah%aquinas@uxc.cso.uiuc.edu

ABSTRACT

In this paper we provide an overview of the current research and development directions in knowledge engineering. We classify research problems in this area according to three attributes: (a) completeness of knowledge in the environment, (b) accuracy of knowledge available in the environment, and (c) knowledge about the objective and/or specifications of the problem. We discuss various approaches to solving problems in this area. Finally, we examine challenges that lie ahead.

I. INTRODUCTION

Intelligent computers are ones that can imitate the intelligent behavior of humans. Studies in this area can be traced back to the study of mathematical logic since the end of the 19th century, when the relation between computation and intelligence was begun to be understood. The seminal work of Church, Turing, and others, show the link between the formalization of reasoning and the computing systems to be invented. Symbols were found to be an integral part of any reasoning system, while numbers were just one way of interpreting the internal states of a computing system.

Not long after the development of the computing machines by Babbage and others, people started to write programs with artificial intelligence, such as ones that play chess, solve puzzles, and translate texts from one language into another. The development of the programming language IPL in the 1950s by Newell, Shaw, and Simon [7] was a pioneering effort of symbolic processing by computers. Data structures of unpredictable shape and size could be manipulated conveniently by programs written in IPL. Many of the early symbolic programs, including the Logic Theorist and the General Problem Solver, were written in IPL. The invention of Lisp in 1958 by John McCarthy further enhanced some of the programming tasks for symbolic processing. The language featured the use of conditional expressions and recursion, representation of symbolic information externally by lists and internally by linked lists, and representation of program and data using the same data structures [5].

A computer can execute instructions at the rate of millions of steps per second. However, it is not considered intelligent unless it can imitate the process of intelligent thinking in humans. Consider a computer program that plays chess. A good one can probably play as well as a human chess master. Although thousands of moves are searched by the computer, the crucial point that makes the chess program proficient lies in the knowledge captured in the program, which guides the search in a way a human chess player does.

According to Websters dictionary [10], data refers to numerical information suitable for computer processing, while knowledge refers to the sum or range of what has been perceived, discovered, or learned. Knowledge can be considered data at a high level of abstraction and can be processed by a computer when it is represented as data. The distinction between these two concepts in terms of computer processing is usually vague [1, 12]. For example, a statement that data engineering is a growing field is a piece of knowledge. This statement has to be substantiated by facts (or data) or by algorithms (or programs) that can generate the supporting evidence. Since the amount of data necessary to characterize a piece of knowledge can be infinite in size, so the meaning of a piece of knowledge can be imprecise or uncertain. In general, knowledge can be considered as a compact and sometimes imprecise way of representing a body of data.

A computer system can be considered logically as an implementation of the knowledge supplied by its

Research supported by National Aeronautics and Space Administration Contract NCC 2-481 and National Science Foundation Grant MIP 88-10584.

Proc. IEEE TENCON'90, IEEE Region 10 Conference on Computer and Communication Systems, Hong Kong.

designers. It can be modeled as a system which consists of a core element, an application-dependent element, and an input/output interface to the environment. The core element is invariant in the application and is provided by the designers. In a conventional system running an application program, the core element consists of algorithms and data structures, whereas the application-dependent element has data and knowledge. Both symbols and numbers are used to communicate with such a system. In contrast, in a symbolic computer, the core element consists of the basic algorithms and necessary data structures, which are used to manipulate data, knowledge, algorithms and data structures stored in the application-dependent element. A computer that learns has a learning program in its core element, which can modify knowledge represented in programs and data structures stored in the application-dependent element.

In general, the core element of a computer system consists of meta-knowledge, which is knowledge about knowledge, and the application-dependent element has domain-knowledge and data. A system that learns has its meta-knowledge implemented as a learning algorithm in the core element. To generalize this further, meta-meta-knowledge can be implemented in the core element, which can manipulate meta-knowledge and domain knowledge stored in the application-dependent element. As a result, as we include higher-level abstract or common-sense knowledge in the core element, the system is capable of higher-level reasoning and becomes "more" intelligent.

As the complexity of applications grow, the need to manage knowledge becomes imminent. Knowledge engineering collectively refers to the methods, algorithms, and systems for the design, utilization, and maintenance of knowledge and data [8,11]. This involves (a) methods for automated acquisition and learning of new knowledge and data, (b) modeling, design, access, control, and evaluation of knowledge engineering systems, (c) representation, language, and architectural supports, and (d) deployment, evolution, maintenance, and standardization of knowledge engineering systems with existing and emerging technologies. In short, knowledge engineering can be considered as studies related to computer-aided management of information, data, and knowledge. The scope of problems involved in this area are continuously evolving, as new applications arise and emerging technologies become mature.

The early years of computing research were dominated by information processing. Loosely speaking, information refers to bits that are stored in computer memories. These bits include data, software, and knowledge represented in data and software. In von Neumann computers, for example, data and programs are stored in the same memory areas. Therefore, data can sometimes be interpreted as programs, while programs are sometimes considered as data. Since knowledge can be treated as a general class of data, its characterization may be uncertain or imprecise. Knowledge can be represented in computers as data or software, or as a mixture of both.

With the growing complexity of applications, it was soon discovered that techniques for designing and processing software were quite different from techniques for processing data, and that techniques for acquiring and managing knowledge were different than techniques for data processing and information processing. This discovery led to the development of structured programming, database processing [9], and artificial intelligence [3,4] in the 1960s and 1970s. Early data and knowledge processing systems were small in scale: modeling, design and management could be handled by one or a few experts.

The ever increasing complexity of applications and information processing systems in the 1980s and the increasing need to capture and manage abstract knowledge require the collective effort of a large number of experts. Data and knowledge can no longer be handled by individuals alone, and its management must be treated as an engineering discipline.

Knowledge engineering systems are feasible because of recent advances in technology and computer architecture. Existing technologies, such as VLSI, VHSIC and fiber optics; and emerging technologies, such as lightwave technologies, sea of gates, three-dimensional VLSI and superconductivity, promise faster computers, more memory, and higher networking bandwidth. Research on multiprocessing and distributed processing also allow faster and parallel computers to be utilized efficiently.

Studies on knowledge engineering, therefore, span a broad spectrum of areas and encompass data, information, knowledge, technology, and products. The subject area is truly dynamic and its emphasis may changes as new applications evolve and new technologies become mature.

In this paper we provide an overview of the current research and development directions in knowledge engineering. We classify research problems and approaches in this area, and discuss future trends. We do not attempt to survey all related work and references in the area, as the scope of work in this area is extremely broad. Further, since knowledge engineering has been applied in many areas, it is not possible to provide a complete discussion of each application.

2. CHARACTERIZATION OF KNOWLEDGE ENGINEERING PROBLEMS

Solutions to problems in knowledge engineering applications depend on the characteristics of these problems. In this section, problems are classified according to three attributes: (a) completeness of knowledge in the environment, (b) accuracy of knowledge available in the environment, and (c) knowledge about the objective and/or specifications of the problem. Table 1 shows the eight possible combi-

Table 1. Characterization of knowledge engineering problems.

COMPLETENESS	EXACTNESS	OBJECTIVE/ SPECIFICATIONS	EXAMPLES	PROCESSING TECHNIQUES
Complete	Exact	Well-defined	Querying a relational database	Parallel processing, combinatorial searches
		Ill-defined	Querying a statistical database	Heuristics must first be applied to define objective/specifications. Learning algorithms is one possible heuristic.
	Inexact	Well-defined	Finding the youngest person in a group; Recognizing an object in a given image	Heuristics must first be applied to find better data/knowledge or restrict data to a fixed degree of precision.
		Ill-defined	Recognizing language in a voice pattern	See comment above for ill-defined objective/ specifications and inexact data.
Incomplete/ Unbounded	Exact	Well-defined	Proving a theorem	Heuristics must first be applied to find a complete set of data/knowledge. If this is not possible, heuristics must be used to define a restricted set of knowledge/data that can be used in solving the problem.
		Ill-defined	Deciding where to process a task to minimize the average response time	See comments above for incomplete data and ill-defined objective/ specifications.
	Inexact	Well-defined	Finding points with the lowest energy in a bounded space	See comments above for incomplete and inexact data.
		Ill-defined	Predicting changes in weather; Reasoning with fuzzy logic	See comments above for incomplete and inexact data and ill-defined objective/specifications.

nations of these three attributes and offers examples for each class. Possible techniques in solving problems in each class are also indicated.

Knowledge/data available in the environment can be complete or incomplete. When it is complete, no additional knowledge is needed to solve the problem. In contrast, when it is incomplete, heuristics must first be applied to find a complete set of knowledge/data. In some cases, the amount of knowledge may be very large or unbounded, and heuristics must be applied to define a restricted set of knowledge/data that can be used in solving the problem. For example, in proving a theorem, it may not be possible to define all of the necessary axioms. As a result, the original problem has to be heuristically modified to prove the theorem based on the *available* axioms.

The knowledge/data available in the environment may be exact or inexact. When it is exact, it can be represented in numerical or logical form. Data/knowledge is inexact when the number of possible cases is infinite, and it is impossible to enumerate or represent all of them. In such cases, before the problem can be solved, heuristics must first be applied to either define a finite number of possibilities or redefine the meaning of exactness so that what is available can be treated as exact.

For example, the birthdays of people in a group are points in a time spectrum, which must first be converted into real numbers of finite precision before the youngest person can be found. In a second example, recognizing an object in a given image involves incomplete data because there are an infinite number of possible orientations and features of the object concerned. To solve the problem, it is necessary to identify a finite and reasonable number of features of the object and heuristically match them with those found in the image. As another example, finding points with the lowest energy in a bounded amount of space is a problem with incomplete and inexact data. Although there is a finite number of particles in such a space, it is not possible to examine the energy value of each. Measurements of a finite degree of precision will be made for a reasonable number of points, and interpolations will be made for intermediate points. Heuristics are, therefore, applied to redefine the meaning of exactness and completeness in the original problem.

After knowledge/data used in solving the problem is defined or restricted, the problem can be solved by finding a solution in the given solution space. This can be represented as the objective of what the solutions must achieve and the specifications that the solutions must satisfy.

An objective of a problem may be well-defined or ill-defined. A well-defined objective can be represented exactly in terms of the measurable parameters of the problem environment so that it is possible to compare the quality of one solution with another. An ill-defined objective may involve either parameters that cannot be expressed in measurable terms or an unknown relationship among measurable parameters. As a result of these conditions, it is not possible to compare the quality of alternative solutions. An ill-defined objective must first be heuristically transformed into a well-defined one before the problem can be solved. This may involve restricting consideration to measurable parameters and defining the objective as a function of these parameters.

The specifications that the solution must satisfy may also be either well-defined or ill-defined. In a well-defined set of specifications, it is possible to enumerate all candidate solutions. In an ill-defined set of specifications, a potentially infinite solution space is defined. Problems with ill-defined specifications must first be heuristically transformed so that all specifications are defined precisely in terms of measurable parameters of the environment.

The querying of a statistical database is an example of a problem with an ill-defined objective because one can only obtain probabilistic rather than precise answers to queries. The recognition of natural language from a voice pattern is a problem with inexact data and ill-defined objective because the inputs are represented as analog signals that are prone to errors, and because there are an infinite number of possible pronunciations of a word. In these conditions, the specifications of the solution space cannot be defined precisely. For this problem, only a finite number of possible pronunciations can be tested. The decision of where a task should be performed in a distributed computer system is another problem with an ill-defined objective because the relationship between the objective (minimizing the response time) and the measurable parameters (workload statistics) is unknown. A heuristic function relating the response time and the measurable quantities must first be defined before the problem can be solved. Finally, the prediction of weather is a problem which may involve an infinite amount of inexact input data and ill-defined objectives and specifications.

3. RESEARCH ON KNOWLEDGE ENGINEERING

The goal of research in knowledge engineering is to study and design systems to support knowledge engineering applications. Approaches to achieving this goal range from theoretical analysis, mathematical modeling, simulations and prototyping, to complete system integration.

The design process of a knowledge and data engineering system is extremely complex and iterative. It is not possible to classify all of the variations of this process without over-simplifying it. However, the resulting design has notable characteristics depending on its starting point, which usually dictates the nature of the final result and can be used as a classification scheme for results in this area. The starting point can be either application-driven or technology-driven.

In an application-driven or a top-down approach, research is focused on first identifying and refining the requirements of the problem. The knowledge necessary for solving the problem is then acquired, and algorithms are designed and mapped to physical systems. In this approach, the capabilities of the resulting system are a good match for the requirements of the application. However, in some instances the requirements are specified in such as a way that a system cannot be physically realized. In this case the design process has to be iterated through repeated refinement of the requirements and designs.

In contrast, a technology-driven or a bottom-up approach determines technological capabilities and limitations first. These limitations lead to the design of realizable systems on which the applications can be mapped. Tradeoffs on performance are then evaluated, and one of the candidates is selected as the target system. The problem with this approach is that the resulting design may not be a perfect match for the application requirements. The design process has to be iterated by modifying the design until the application requirements are satisfied.

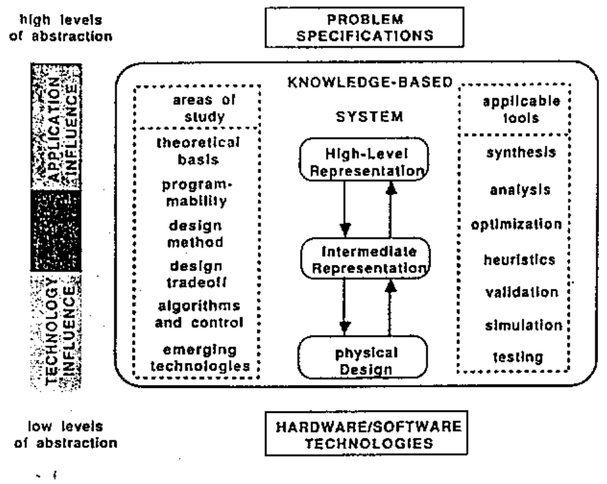

Figure 1. Design of knowledge-based systems.

Research on knowledge engineering addresses the theory, analysis, design, development, evaluation, and maintenance of new knowledge management techniques, methodologies and systems that can support specialized applications in a cost-effective manner. It also emphasizes a better understanding of problem solving methods with respect to the applications concerned. Related issues in accomplishing these goals include studies on the theoretical aspects, design tools and methodologies, design tradeoffs, representation and programmability, algorithms and control, reliability and fault tolerance, and designs using existing and emerging technologies. In the following text, we discuss each of these issues by stating its goals and showing some representative examples. It must be pointed out that all of these issues must be considered in designing a complete working system.

Figure 1 depicts a hierarchical organization of studies in knowledge engineering. The problem to be solved may be specified in an imprecise form and must be successively refined into a form that can be implemented physically. Applicable methods for such refinement include synthesis, analysis, optimization, verification, validation, testing, and maintenance. Representations on a higher level are more influenced by the applications, while representations on a lower level are influenced by technological constraints.

Theory. Theoretical studies involve abstracting the problem domain and studying the properties and limitations of the abstract problem. Such an abstraction is often necessary to transform the original problem, which is too complex to be studied in its entirety, into a simpler and more manageable form. Moreover, since the abstract model may cover a number of different problems in the class, the results developed will be applicable to these problems as well. Theoretical results can be used to guide the selection of algorithms and design tradeoffs.

Programmability and Representation. This issue involves choosing or designing the appropriate knowledge and data representation schemes and developing the necessary hardware and software support mechanisms. A choice must be made between using an existing representation scheme, such as one that is declarative, procedural, functional or object-oriented; or designing a new representation scheme. The scheme may have to take into account uncertain, fuzzy or incomplete knowledge and data in the application, and may need to support data and knowledge engineering applications in a sequential or parallel environment.

Many of the issues of representation should be considered early in the design process when the problem semantics are still available. An essential issue to consider is specifying a minimal amount of information to allow control to be carried out efficiently in an implementation. This knowledge should preferably be specified in a form that is independent of the computer architecture on which the knowledge management system is implemented. Detection of this information for efficient control may have to be delayed until run time when more accurate knowledge about the resources is available.

Design Tools and Methodologies. The design of a knowledge-based system is a complex task, which can be greatly simplified by using computer-aided design tools. Systematic methods must be used for analyzing requirements, representing specifications, partitioning problems into a manageable size, designing algorithms, mapping algorithms into physical knowledge management subsystems, and maintenance of these systems. The process is iterative and stops when problem requirements, such as flexibility, extendibility, reconfigurability, user friendliness, reliability and evolvability; and technological limitations, such as chip size, bandwidth and packaging, are met.

Design Tradeoffs. Design tradeoffs for knowledge management systems involve tradeoffs on hardware and software implementations, representation schemes, use of software languages, computational power of processors, communication bandwidth, storage capacity, cost, design of specialized functional units, operating system environment, and use of hardware technologies. A judicious selection of the components to be used in the system must be made in order to obtain a high performance system with acceptable cost.

Algorithms and Control. This issue involves the design and evaluation of algorithms and control strategies for efficient and reliable operations in knowledge management systems. It also involves methods to acquire, test, store, access, and maintain reliable data and knowledge for a given application.

The design process is usually hampered by imprecise or incomplete knowledge at design time, which necessitates developing and using heuristics for control. These heuristics may underutilize resources and cause anomalous behavior with respect to resources. In the latter case, increasing the resources available to a system may result in worse performance by the heuristics (with respect to the objective).

Another problem in the design of algorithms is that they depend largely on the characteristics of the application involved and the experience of the designers. Automated methods for designing and evaluating algorithms are not available at this time. Computer aided tools, such as automated learning methods, may be very useful in adapting existing algorithms to new applications.

Emerging Technologies. This issue involves exploration of the limitations and impact of emerging technologies on the design, evaluation, programmability, and application span of knowledge engineering systems. Emerging technologies, such as optical processing [11], artificial neural networks [2], high bandwidth optical links, new packaging technologies [6], large memory devices and new architectural concepts, may impact the ways that design tradeoffs are performed and algorithms are developed. They may also result in different design methods, problem solving strategies, and control algorithms for future knowledge engineering systems and impact the evolution of current systems.

Table 2 shows a few examples of some of the issues that are studied in knowledge engineering. Some of these issues are more exploratory in nature, while others may be more related to industrial practices.

4. FUTURE CHALLENGES

Improvements in knowledge engineering systems can come from faster technologies, better representation schemes, more efficient algorithms, automated software design methods, and better hardware/software architectures integrated with the available technologies.

Emerging Technologies. The basis for any computer system is the technology in which it is implemented. The design of a system is often driven by its cost, hence, the fastest technologies, subject to cost specifications, are used. New technologies may give higher performance but are often prohibitively expensive. These improvements will likely bring a two to three orders of magnitude improvement in computational speeds in the next decade.

Knowledge Representations. Many new representation schemes have evolved in the recent past. These schemes may feature tools for knowledge and data capture and management. However, they are usually not directed towards any specific applications and may have to be modified or extended to tailor to the applications and computational environment. Another major problem is the lack of an overall technique to guide the evaluation and selection of a representation scheme. Research in this area could prove extremely valuable. Learning techniques for incorporating new knowledge about application

Table 2. Examples of knowledge engineering issues.

Theoretical Basis
Learning theory
Database models
Recursive queries
Non-monotonic knowledge base
Logic databases and problem solving
Programmability and Representation
Truth maintenance
Object-oriented models
Query, design, and visualization languages
Knowledge and data representation schemes
Design Methods and Tools
Life-cycle maintenance
Computer-aided design tools
Knowledge acquisition schemes
System integration and modeling
Statistical databases and knowledge bases
Design Tradeoffs
Standardization
Application experience
Real-time knowledge bases and databases
Performance evaluation techniques and tools
Integrated voice, image, and data processing
Intensional versus extensional data processing
Algorithms and Control
Search algorithms
Garbage collection
Hashing techniques
Resource scheduling
Integrity maintenance
Data communication aspects
Machine learning algorithms
Parallel and distributed processing
Distributed virtual memory management
Fault detection, checkpointing and recovery
Emerging Technologies
Optical processing
High-speed optical links
Artificial neural networks
Knowledge-base coprocessors

domains into current solutions in a knowledge-intensive application may also impact knowledge engineering.

Algorithms. Research in the area of application-specific algorithms will have the greatest potential for speeding the solution of the given application. The development of new and improved algorithms for an application can be seen as finding alternative ways to incorporate knowledge about the application domain and the technology into the computer solution.

The design of better algorithms and knowledge representation schemes is an important complement to the tremendous potential offered by emerging technologies. Extension in the limit of processing power by new technologies is valuable, especially for real-time systems. However, the two to three orders of magnitude speed improvement offered by these technologies in the next decade will not greatly impact the size or type of knowledge engineering applications that are addressed today. Many of these applications involve huge search spaces of an exponential size; two to three orders of magnitude increase in computational speed will do little to extend the size of a solvable instance of such a problem [9].

In general, faster and parallel computers are useful in improving the turnaround time of algorithms which can already be evaluated on existing computer systems. In the near future, faster computers alone will not likely make a significant difference in solving very complex problems that we cannot solve today. Better methods of harnessing knowledge (which result in better representation methods and algorithms), together with faster computers, will allow us to extend the spectrum of problems solved today.

Software Architecture. Software architectures is an integral part of a knowledge management system. Generation of new software environments, tools, and languages will probably rely on amalgamation of known representation techniques and software design methodologies. Software development systems and automated intelligent assistants represent prime areas for advancement of knowledge engineering applications. The problems of verification and validation and continuous maintenance of programs, knowledge and data are important related topics.

Hardware Architectures. As with software, hardware architectures are often based on known design techniques such as parallel processing and pipelining. Innovation for new architectural concepts may be made possible by new and emerging technologies, which affect the cost effectiveness of previous designs.

New hardware architectures are best utilized for operations that the computer performs frequently. Counter to intuition, identification of these tasks is very difficult. Operations may be instructions, parts of instructions, groups of instructions, or frequently recurring tasks. Identification of new and valid areas for development of new hardware architectures is an important area of research.

System Design. System-level design is often based on an overall design philosophy and may contain, for example, a mix of different computational models, and general-purpose and special-purpose functional units. An important driving force is the cost-effectiveness of the design and its technological feasibility. A major difficulty, however, lies in integrating designs with different knowledge representation schemes, software architectures and hardware components, and evolving systems as technologies change.

REFERENCES

[1] "Special Issue on Data Engineering," *IEEE Computer*, vol. 19, no. 1, Jan. 1986.

[2] DARPA, *DARPA Neural Network Study*, Lincoln Laboratory, Massachusetts Institute of Technology, Lexington, MA, July 1988.

[3] A. Barr and E. A. Feigenbaum, *The Handbook of Artificial Intelligence*, vol. 1, 2, and 3, William Kaufmann, Los Altos, CA, 1981, 1982.

[4] J. McCarthy, "Program with Common Sense," in *Mechanization of Thought Processes*, pp. 75-84, Her Majesty's Stationery Office, London, 1959.

[5] J. McCarthy, "History of Lisp," *SIGPLAN Notices*, vol. 13, no. 8, pp. 217-223, ACM, 1978.

[6] J. F. McDonald, H. J. Greub, R. H. Steinvorth, B. J. Donlan, and A. S. Bergendahl, "Wafer Scale Interconnections for GaAs Packaging--Applications to RISC Architecture," *Computer*, vol. 20, No. 4, pp. 21-35, IEEE, Apr. 1987.

[7] A. Newell, J. C. Shaw, and H. A. Simon, "Programming the Logic Theory Machine," *Prof. 1957 Western Joint Computer Conf.*, pp. 230-240, IRE, 1957.

[8] C. V. Ramamoorthy and B. W. Wah, "Knowledge and Data Engineering," *Trans. on Knowledge and Data Engineering*, vol. 1, no. 1, pp. 9-16, IEEE, March 1989.

[9] J.D. Ullman, *Principles of Database Systems*, pp. 211-267, Computer Science Press, 1984.

[10] B. W. Wah, G.-J. Li, and C.-F. Yu, "Multiprocessing of Combinatorial Search Problems," *Computer*, vol. 18, no. 6, pp. 93-108, IEEE, 1985.

[11] B. W. Wah, "Knowledge and Data Engineering," in *Encyclopedia of Computer Science and Technology*, ed. A. Kent and J. G. Williams, Marcel Dekker Inc., New York, NY, 1990.

[12] H. Webster, *Webster's II New Riverside University Dictionary*, Riverside Pub. Co., Boston, MA, 1984.

[13] L. C. West, "Picosecond Integrated Optical Logic," *Computer*, vol. 20, pp. 34-47, IEEE, Dec. 1987.

[14] G. Wiederhold, "Knowledge and Database Management," *Software*, vol. 1, no. 1, pp. 63-73, IEEE, Jan. 1984.