DECEMBER 1983 VOL. 6 NO. 4

a quarterly bulletin
of the IEEE computer society
technical committee
on

Database Engineering

Contents

Database Engineering Bulletin is a quarterly publication of the IEEE Computer Society Technical Committee on Database Engineering. Its scope of interest includes: data structures and models, access strategies, access control techniques, database architecture, database machines, intelligent front ends, mass storage for very large databases, distributed database systems and techniques, database software design and implementation, database utilities, database security and related areas.

Contribution to the Bulletin is hereby solicited. News items, letters, technical papers, book reviews, meeting previews, summaries, case studies, etc., should be sent to the Editor. All letters to the Editor will be considered for publication unless accompanied by a request to the contrary. Technical papers are unrefereed.

Opinions expressed in contributions are those of the individual author rather than the official position of the TC on Database Engineering, the IEEE Computer Society, or organizations with which the author may be affiliated.

Membership in the Database Engineering Technical Committee is open to individuals who demonstrate willingness to actively participate in the various activities of the TC. A member of the IEEE Computer Society may join the TC as a full member. A non-member of the Computer Society may join as a participating member, with approval from at least one officer of the TC. Both a full member and a participating member of the TC is entitled to receive the quarterly bulletin of the TC free of charge, until further notice.

Membership applications and requests for back issues should be sent to IEEE Computer Society, P.O. Box 639. Silver Spring, MD 20901. Papers and comments on the technical contents of Database Engineering should be directed to any of the editors.

Letter from the Chief-Editor

This issue is about expert systems and database systems. As the issue focuses on a topic which straddles two different disciplines, it has turned into the most voluminous issue in the history of Database Engineering. An expert system is generally defined to be a computer program which uses explicitly represented knowledge and inference procedures to solve problems in some specific domain which have traditionally depended on human experts. As I suspect many of the readers of Database Engineering may not be familiar with expert systems, I wanted this issue to first of all serve as a brief introduction to expert systems and then to indicate how database systems may be made more intelligent by either connecting them with expert systems or by augmenting them with some expert system techniques.

The first three papers are intended to introduce some fundamental aspects of expert (knowledge-based) systems. The "Short Introduction to Expert Systems" by Clifford, Jarke and Vassiliou provides an overview of the architecture of expert systems and outlines directions of current research. In "Knowledge Engineering and Fifth Generation Computers," Furukawa and Fuchi, leaders of Japan's 5th generation computer project, provide a rationale for their choosing a Prolog-based logic programming language as the basis of the next generation of computer systems and then give an overview of the Fifth Generation Computer project in Japan. Parsaye gives a brief introduction to the Prolog language and indicates the relationship between Prolog and relational databases in "Logic Programming and Relational Databases."

The next three papers were selected to provide some concrete examples of expert systems, including two of the best-known expert systems currently in use, R1 and Prospector. McDermott, in "The Knowledge Engineering Process," first discusses the steps involved in constructing expert systems and then relates them to the R1 computer configuration assistant system. In "A Review of the Prospector Project," Reiter provides a summary of the outstanding features of the Prospector geological exploration assistant system and discusses what has been learned from the project. "An Orthodontic Case Study Instruction System Based on a Relational Database System," by Kanamori, Sugawara, and Masunaga, describes an instructional system which, due to the domain of expertise it addresses, does not have the conventional inference capabilities. This is a non-invited paper.

The question of how database systems may be made more intelligent is addressed in the next four papers. Vassiliou, Jarke, and Clifford outline a research project in expert systems and describes some architectural alternatives in bridging expert systems with database systems. In "Basic Decisions about Linking an Expert System with a DBMS: A Case Study," Lafue cautions that the choice of an implementation language for expert systems depends on applications and that sometimes it may be better to extend expert systems into database management systems. Then Stonebraker, Woodfill and Andersen describe how an existing relational database system may be made more intelligent by incorporating a rules system. Another proposal for making database systems more intelligent is to make use of the semantic knowledge about the data. The results of this line of research are summarized in "Applications of Artificial Intelligence in the Knowledge-Based Management Systems Project" by Wiederhold, Milton, and Sagalowicz.

It has been a pleasure to work with the authors who contributed to this issue. The cooperation I have received from them, who I realize are horrendously busy, has been truly gratifying. I owe special thanks to Dr. Adrian Walker, Prof. Yannis Vassiliou, Prof. Gio Wiederhold, and Prof. Michael Stonebraker for providing me with leads to the authors who contributed papers to this issue.

Our publication schedule for 1984 is as follows. Don Batory will open 1984 with an issue on Statistical Databases. Randy Katz will follow with an issue on Engineering Design Databases. Then Dave Reiner will do an issue on Operating System support for Database Systems.

Won Kim

October, 1983
San Jose, Calif.

A SHORT INTRODUCTION TO EXPERT SYSTEMS

Jim Clifford, Matthias Jarke, and Yannis Vassiliou

Computer Applications and Information Systems Area
Graduate School of Business Administration
New York University

It is a generally accepted view among researchers in Artificial Intelligence that the 1980's will witness a tremendous upsurge in the number of successful applications of AI expertise to real-world systems. High on the list of the technologies that are expected to be applied in the marketplace are expert, or knowledge-based, systems. The formation of a number of expert system companies, often in close collaboration with major academic AI research centers, attests to the growing belief in the economic viability of this technology transfer. Although there is yet to be developed a formal theory of what constitutes an expert system, there are some general features that can be identified.

An expert system (ES), by definition, is a computer system which attempts to act like a human expert in some limited application domain. For decades people have certainly been building computer systems that have attempted to be expert in their field of application -- no one has purposefully (unless maliciously) built a system that was intended to bungle its job! There are perhaps two aspects to an expert system that distinguish it from more traditional computer systems: overall architecture, and method of development.

An expert system architecture consists of two interacting components: a "knowledge base" and an "inference engine." The knowledge base contains all of the information that a human expert would normally need to carry out the desired task. This knowledge base itself is usually divided into two sub-components, the first containing specific, or "ground" facts (e.g., "Mary Smith is 35 years old"), and the second containing more general principles, rules, or problem-solving heuristics (e.g., "If a person is single then that person has no spouse"), which come from accumulated empirical observations or technical knowledge of the domain. An important feature of ES's is that both of these knowledge bases are stored declaratively in some assertion language, and not buried somewhere in computer code. This means that the knowledge incorporated into the system is easily accessed by the users, and potentially more easily modified or extended. The second component in an ES is a general purpose inference engine that is capable of making decisions from, answering questions about, and determining the consequences implied by the knowledge that is built into the system.

The other unusual aspect of expert systems is the manner in which they are constructed. The architecture of an ES in a way dictates the often-quoted motto of ES researchers that "in the knowledge lies the power." What this slogan means is that the knowledge base component of an ES contains <u>all</u> of the domain-specific information for the application. In practice, because of the declarative nature of this knowledge base, and the power of the AI languages that have been developed for these systems, this has led to an incremental approach to ES development. Working in small teams of about 3 people, consisting minimally of the domain expert, a programmer, and a knowledge engineer, a small prototype ES is developed, usually in a matter of 2 or 3 months. The system is then successively refined in a process of examining its behavior, comparing it to that of the human expert, and correcting its reasoning processes by modifying its knowledge base. This process continues until the system performs at a level of expertise that approximates that of the human expert. At this point the system is ready for evaluation in the field. However, just as a human expert never stops developing or expanding his/her expertise, the ES is structured to facilitate continued growth and expansion of its capabilities.

In this short paper, some basic aspects of the structure and range of expert system applications are addressed, and directions of current research are indicated. Other comprehensive references on the subject are: [Barr and Feigenbaum 1982, Buchanan 1981, Davis 1982, Duda 1981, Gevarter 1982, Hart 1982, Hayes-Roth 1981, Hayes-Roth et al 1983, Michie 1980, Nau 1983, Stefik et al 1982].

1.0 <u>ARCHITECTURE OF EXPERT SYSTEMS</u>

For a long time, artificial intelligence has concentrated on the development of procedural techniques and representations such as heuristic search methods and problem transformation techniques. These have proven too general to solve real world problems in specific domains. Therefore, the focus has shifted to the representation and use of <u>domain knowledge</u> to guide search processes more efficiently.

The observation that human domain experts use domain knowledge as well as meta-knowledge (knowledge about the scope of one's knowledge and knowledge about how to use one's knowledge) efficiently has led to the idea of extracting knowledge from a human expert into a <u>knowledge base</u>. The knowledge base is therefore at the heart of any expert system. It is a storehouse of knowledge in the form of specific facts and general rules, or in frames of reference that structure the expert's experience and expectations.

To exploit the knowledge, an <u>inference engine</u> is required that relates a problem description to the stored knowledge in order to <u>analyze</u> a certain situation (e.g., in medical diagnosis) or to <u>synthesize</u> a solution for a specific problem (e.g., a computer configuration). Such an inference engine can be a pattern matcher, theorem prover, or network search mechanism customized for one expert system, or it may exist already in the compiler of a corresponding knowledge representation language such as OPS-5 [Forgy 1980], Prolog [Kowalski 1979], or EMYCIN [van Melle 1979]). Even in the latter case, some additional control mechanism may be required to cut down the number of inferences to be made.

The third major component of an expert system contains a number of user interfaces for various purposes. The two most important seem to be an interface for knowledge acquisition through which the expert or an intermediary can insert, update, and check knowledge in the knowledge base, and an interface through which end-users can get consultation from the expert system. As a windfall profit, the stored expertise can sometimes be made available to train new human experts.

2.0 KNOWLEDGE REPRESENTATION AND INFERENCE PROCEDURES

The knowledge base may require the description of facts about specific objects, relationships, and activities; of classification and generalization hierarchies; of general relationships between object and activity classes; and of meta-knowledge about the scope, importance, precision, and reliability of the stored knowledge. Just as database research has developed multiple representations for specific facts, many techniques exist to represent the more general knowledge required for expert systems.

A "good" knowledge representation should support the tasks of acquiring and retrieving knowledge as well as of reasoning. Factors that have to be taken into account in evaluating knowledge representations for these three tasks include:

1. the naturalness, uniformity, and understandability of the representation;

2. the degree to which knowledge is explicit (declarative) or embedded in procedural code;

3. the modularity and flexibility of the knowledge base;

4. the efficiency of knowledge retrieval and the heuristic power of the inference procedure (heuristic power is defined as the reduction of the search space achieved by a mechanism).

Below, four major knowledge representation techniques and their related inference mechanisms will be briefly reviewed. A thorough examination of knowledge representation is given in [Mylopoulos 1980].

2.1 Production Rules

Rules [Davis, Buchanan, and Shortliffe 1977] have been the most popular form of knowledge representation in expert systems. [Chandrasekaran 1983] points out three interpretations of the function of rules in expert systems. First is the interpretation of rules as a programming language. A rule typically has the form

if X then Y.

It can be used in computations in different ways. On one hand, in a data-driven or forward chaining approach, one can try to match a given situation to the condition X in order to infer a possible action Y. On the other hand, one can try to "prove" a hypothesis Y by establishing the preconditions X through further analysis (backward chaining). Combinations of both methods are also sometimes used.

Both approaches require a pattern matching process, perhaps combined with unification (substitute constants or other variables for variables in the pattern to be matched) to identify the applicable rules in a given problem situation. If there is more than one of those, one has to be selected for further processing first. Control structures for rule application can be distinguished by their flexibility of rule choice into irrevocable ("hill-climbing") or tentative, and by the sequence of analysis in depth-first with backtracking or breadth-first with parallel graph search [Nilsson 1980].

Secondly, rules can be used as description tools for problem-solving heuristics, replacing a more formal analysis of the problem. In this sense the rules are thought of as "rules of thumb," incomplete but very useful guides to make decisions that cut down the size of the problem space being explored. These rules are input to an expert system by the human expert, usually iteratively and perhaps by means of an interactive program that guides and prompts the expert to make this task easier, and perhaps does some limited consistency checking.

Finally, rules have been proposed as in some sense a simulation of the cognitive behavior of human experts. By this claim, rules are not just a neat formalism to represent expert knowledge in a computer but rather a model of actual human behavior.

A problem with rule-based techniques is the organization of the stored knowledge in a way that permits efficient yet transparent control over the search processes inside the knowledge base. There is currently no satisfactory formal solution to this problem but a number of ad-hoc programming tricks have been developed.

2.2 First-Order Logic

Precise knowledge can be stated as assertions over objects that take the form of first-order predicates with functions and equality [Kowalski 1979]. Logic has the advantage of offering a sound and complete set of inference rules. It is also purely declarative and therefore allows multiple uses of the same piece of knowledge. For inference purposes, predicates are usually transformed in a quantifier-free normal form called clausal form.

As an illustration, Prolog's [McDermott 1980, van Emden and Kowalski 1976] inference procedure is based on the resolution principle [Robinson 1965]. In order to prove a theorem in clausal form, its negation is added to the set of knowledge clauses or "axioms". If the thus augmented conjunction of clauses can be shown to be contradictory, the theorem has been proved.

A major problem with general first-order logic as a knowledge representation is again the difficulty to express control structures that efficiently guide the use of a large knowledge base. To reduce such problems, practical tools such as the logic programming language Prolog use the subset of definite (Horn) clauses rather than full first-order logic. Furthermore, these clauses are interpreted in a procedural way similar to backward chaining in production rules, leading to a more efficient search process while reducing somewhat the generality of interpretation possible in a nonprocedural interpretation.

2.3 Networks

Semantic networks [Quillian 1968, Brachman 1979, Schubert 1976] seem to be more popular in other AI applications (e.g., natural language processing) than in expert systems. Nevertheless, a number of expert systems rely on network formalisms, among them very large systems such as INTERNIST [Pople 1983], Prospector [Hart et al 1979], and SOPHIE [Brown et al 1981]. Networks are a natural and efficient way to organize knowledge. Nodes describe objects, concepts, or situations whereas arcs define the relevant relationships. Reasoning corresponds to network traversals along the arcs or to pattern matching of problem descriptions and subnets. A large number of exact and heuristic mechanisms exist for these tasks. The disadvantages of this approach stem from the lack of formal semantics making verification of the correctness of reasoning very difficult.

2.4 Frames

Much knowledge is based on experience and expectations adapted from previous situations and general concepts to a specific problem. Frames [Minsky 1977, Schank 1972, 1975, Bobrow 1977] provide a structure to such experiential knowledge by offering so-called slots which can be filled with type descriptions, default values, attached procedures, etc. Frames are a very general and powerful representation form. It may be difficult, however, to specify their meaning precisely as well as to implement them efficiently.

2.5 Multiple Knowledge Representations

It should be clear by now that no one of the knowledge representation methods is ideally suited for all tasks. In very complex systems using many sources of knowledge simultaneously (e.g., speech recognition [Erman et al. 1980]), the goal of uniformity may have to be sacrificed in favor of exploiting the benefits of multiple knowledge representations each tailored to a different subtask. Similar to the interdisciplinary cooperation of several human experts, the necessity of translating among knowledge representations becomes a problem in such cases.

The need for translation also occurs when an expert system is interfaced with other software systems, e.g. database management systems.

3.0 USER INTERFACES

There are at least three distinct modes of interacting with the expert systems that are now being developed: consultation, knowledge acquisition, and training. Of course not every system allows these three types of interaction, nor is this interaction always facilitated by means of automated tools. Nevertheless the basic expert system architecture that has emerged has shown itself to be capable of at least these modes of interaction. In this section we will give a brief overview of these three interaction types.

3.1 Consultation

The primary mode of interaction is the consultation session, wherein the expert system is used to solve the problem for which it was constructed. There are really two forms that this interaction can take. In the simplest case some member of the user community, not necessarily the expert, presents a problem to the system and requests that the system apply its expertise to generate a solution. Assuming it is capable of understanding the problem statement and then of solving the problem, the system responds to the user with the solution and everyone is happy.

If the user is unhappy with the solution, uncertain as to its validity, or desirous of an explanation of "why" or "how" the system has reached its conclusion, the user can typically enter into a second form of the consultation mode of use and request an explanation of the steps that the system has followed to achieve the generated result. In most cases this explanation takes the form of a formatted presentation of the chain of rules that were activated by the inference engine in reaching the solution. This explanatory capability is a major advantage over more conventional systems, and is facilitated by the architectural feature of a clear separation between the knowledge base and the inference mechanism.

3.2 Knowledge Acquisition

A second form of interaction with the expert system is the knowledge acquisition process, wherein the knowledge and heuristics used by the human expert in the problem-solving task are transferred into the knowledge base of the expert system. This dialogue is the least understood process in the expert system paradigm. In most systems this interaction is not automated, but rather is mediated by a "knowledge engineer" [Feigenbaum 1980] whose job it is to (a) pick the brains of the human expert for the knowledge, principles, and heuristics used to solve the problem at hand, and (b) translate this communicated information into the form(s) required by the representation language(s) within which the expert system is being implemented.

There are very few guidelines available for how to facilitate this process. It is generally recognized that this is a long and tedious process, requiring good conceptual and communication skills, considerable patience, and experience. Moreover, it is this knowledge acquisition process which is iterative, continuing throughout not only the development of the system but during all of its useful life. On the other hand, this is another touted advantage of expert systems over conventionally engineered systems -- the ability to grow and learn, thereby providing the opportunity to continually improve performance. While this expandability is certainly enhanced by the isolation of the knowledge base, it is clearly not always a simple task to expand the limits of a system's expertise.

Much research is currently being devoted to techniques for at least partially automating the knowledge-acquisition process. Such systems as AGE [Nii and Aiello 1979], KAS [Duda et al 1979], TEIRESIAS [Davis and Lenat 1982], EXPERT [Weiss and Kulikowski 1979], HEARSAY-III [Erman et al 1980], etc. have all attempted to provide a framework within which the system can guide the expert in communicating his/her expertise to the system. Much work remains to be done in this area, both in the development of automated tools for the existing paradigms of problem solving, and in the more basic research into the understanding of the very nature of human problem-solving strategies and abilities.

3.4 Training

A final form of interaction with the expert system occurs when the system is used as a training tool to teach new human experts the problem-solving skills embodied in its knowledge base. Relatively few systems have been used in this mode. However, such systems as SOPHIE [Brown et al 1981] have demonstrated that the existence of a clearly formulated, central repository of expertise provides a solid foundation for the development of such computer-based teachers as a fortuitous side-effect.

4.0 CURRENT STATUS - A POTPOURI OF EXPERT SYSTEMS

There is as yet no well-developed theory of problem-solving techniques, no theory of problem space complexity comparable, say, to a theory of database query complexity. Moreover, or perhaps partially in consequence, the development of expert systems is still more of an art than a science. It is therefore difficult to find a concrete opinion held about these systems by a reputable researcher in the field whose opposite is not held by another researcher equally as reputable. Nevertheless, a consensus is beginning to emerge as to the characteristics of problem domains appropriate for the technologies that exist today. Recent surveys of expert systems [Davis 1982, Gevarter 1982, Nau 1983] have emphasized a number of characteristics to look for in a problem domain before considering it as a candidate for current expert system technology, and have identified a number of considerations involved in the development of such a system.

Foremost among these characteristics are the selection of an appropriate domain, and the availability of a human expert. The most successful domains seem to be those wherein the expertise is based on experience of associations, rather than causal links or use of structural information. Equally important is a close collaboration and active participation of the human expert throughout the entire system development process. Other considerations frequently mentioned are: the necessity for an experienced "knowledge engineer," the efficacy of a quick (3 months?) development of a first system prototype to test the feasibility of the initial problem-structuring ideas, and an average development time of 5 years, regardless of the number of people on the project.

In Table 1, examples of expert systems are presented. As can be seen, expert systems have been built for several domains, which include Medicine, Geology, Chemistry and Physics, Mathematics, and Computers (both software and hardware). Among these, R1, Macsyma, and the Dipmeter Advisor are widely used in commercial environments.

Prominent researchers in the area (e.g. [Davis 1982]) see future expert systems departing from simple rules and uniform knowledge representations, to causal models employing multiple representations that concentrate on the understanding and description of "structure" and "function".

4.1 Expert Systems and Database Management

There have been several research efforts to combine expert system technology with that of database management systems. Historically, knowledge-based techniques were first applied at the query language level (e.g., natural language). Systems like RENDEZVOUS, LADDER and KLAUS [Haas and Hendrix 1980] have successfully employed knowledge-bases to disambiguate and process English queries to and about databases. In addition, formal specification languages like TAXIS [Mylopoulos et al 1980] have been proposed for the design of databases and, more generally, information systems. Knowledge-based technology may also be used in such database topics as, query optimization [King 1981], transaction management (e.g., constraint maintenance), and data representation [Jarke and Vassiliou 1983, Vassiliou et al 1983].

A more recent research topic is that of coupling ESs with DBMSs. To date the applications that have been chosen for expert systems have had the property that their knowledge base of rules has been relatively small (around 1000 rules is common) and their base of specific facts has been considerably smaller, usually data pertaining to a single problem case and obtained interactively during system execution. In almost all cases, then, these knowledge bases have been implemented directly in main memory. The work of Kunifuji and Yokota for the Fifth Generation Computer project, and that of Vassiliou, Jarke, and Clifford (accompanying articles in this special issue) attempt to apply the ES paradigm to a problem characterized by the existence of a large database of specific facts which the expert must access in order to perform successfully. These research efforts are based on the logic formalism of Prolog (described in the article of Parsaye of this issue). A critisism of this formalism is presented in the accompanying article by Lafue. In addition, the latter article examines the question of whether an existing database system should be employed for the expert system's database access requirements, or a new system must be built.

SYSTEM	FUNCTION	DOMAIN	REF	SOME UNDERLYING PRINCIPLES
Casnet	Consulting	Medicine	66	Production rules. Causality. Semantic net
Internist	Consulting	Medicine	50	Forward/backward chaining. Frames.
KMS	Consulting	Medicine	52	Conditional probabilities.
MDX	Consulting	Medicine	8	Hierarchical, subproblem formation.
Mycin	Consulting	Medicine	59	Backward chaining. Exhaustive search.
Puff	Consulting	Medicine	33,49	Backward chaining. Exhaustive search.
AQ11	Diagnosis	Plant Diseases	9	Multiple-valued logic.
Dipmeter Advisor	Exploration	Geology	14	Causality.
Prospector	Exploration	Mineral	16	Backward chaining. Semantic net.
R1	Configuration	Computer	31	Forward chaining. No backtracking. Subproblem formation. Pattern match.
EL	Analysis	Circuits	60	Forward chaining. Backtracking. Constraint propogation.
SOPHIE	Troubleshoot	Electronics	5	Multi-knowledge representation.
Molgen	Planning	DNA Exper.	36	Forward/backward chaining. Hierarchical, subproblem formation.
Macsyma	Manipulation	Math	43	Pattern match.
AM	Formation	Math	34	Forward Chaining. Generate, test.
Dendral	Generation of hypotheses	Chemistry	21,35	Forward Chaining. Generate, test.
SYNCHEM2	Organic Synth.	Chemistry	23	Multi-representation. Subproblem formation
Hearsay	Interpretation	Speech Recognition	1	Forward/backward chaining. Multi-representation.
Harpy	Interpretation	Speech Recognition	37	Forward chaining
Crysalis	Interpretation	Crystallo-graphy	17	Event Driven. Generate, test.
Noah	Planning	Robotics	54	Backward chaining. Subproblem formation.
Abstrips	Planning	Robotics	55	Back-chaining. Hier. sub-problem formation
VM	Monitoring	Medicine	19	Event Driven. Exhaustive Search.
Guidon	CAI	Medicine	10	Event Driven.

TABLE 1: Representative Expert Systems

References

1. Balzer, R., Erman, L.D., London, P., and Williams, C., "HEARSAY-III: A Domain-Independent Framework for Expert Systems," Proc. 1st Natl. Conf. of the Amer. Assoc. for Artificial Intelligence, Palo Alto, 1980, pp. 108-110.

2. Barr, A., Feigenbaum, E.A., The Handbook of Artificial Intelligence, Volume 2, William Kaufmann, Inc., Los Altos, CA, 1982.

3. Bobrow, D.G., and Winograd, T. "An Overview of KRL, a Knowledge Representation Language," Cognitive Science, Vol. 1, No. 1, 1977, pp. 84-123.

4. Brachman, R. "On the Epistemological Status of Semantic Networks," Associative Networks: Representation and Use of Knowledge by Computer, N.V. Findler, ed., Academic Press, 1979, pp. 3-50.

5. Brown, J., Burton, R, deKleer, J., "Pedagogical Natural Language and Knowledge Engineering Techniques in SOPHIE I, II, and III", Intelligent Tutoring Systems, Sleeman et al (eds), Academic Press, 1981.

6. Buchanan, B.G., "Research on Expert Systems," Stanford University Computer Science Department, Report No. STAN-CS-81-837, 1981.

7. Chandrasekaran, "Expert Systems: Matching Techniques to Tools", Artificial Intelligence Applications for Business (W.Reitman, ed.), Ablex, to appear 1983.

8. Chandrasekaran et al., "An Approach to Medical Diagnosis Based on Conceptual Structures," Proc. Sixth Int'l Joint Conf. Artificial Intelligence, 1979, pp. 134-142.

9. Chilausky, R., Jacobson, B., and Michalski, R.S. "An Application of Variable-Valued Logic to Inductive Learning of Plant Disease Diagnostic Rules," Proc. Sixth Annual Int'l Symp. Multiple-Valued Logic, 1976. Tokyo, Aug., 20-23, 1979, pp. 645-655.

10. Clancey, W.J., "Dialogue Management for Online Tutorials", IJCAI 6, 1979, pp.155-161.

11. Davis, R., "Expert Systems: Where are we? and Where do we Go from Here?", Massachusetts Institute of Technology, AI MEMO No. 665., June, 1982.

12. Davis, R., and Lenat, D.B. Knowledge-Based Systems in Artificial Intelligence, New York: McGraw-Hill, 1982.

13. Davis, R., Buchanan, B., and Shortliffe, E. "Production Rules as a Representation for a Knowledge-Based Consultation Program," Artificial Intelligence, Vol. 8, No. 1, 1977, pp. 15-45.

14. Davis, R., et al., "The Dipmeter Advisor: Interpretation of Geological Signals," Proc. Seventh Int'l Joint Conf. Artificial Intelligence, 1977, pp. 1030-1037.

15. Duda, R.O., Gaschnig, J.G., "Knowledge-Based Expert Systems Come of Age," Byte, Vol. 6, No. 9, Sept. 81, pp. 238-281.

16. Duda, R.O., Gaschnig, J.G., and Hart, P.E., "Model Design in the PROSPECTOR Consultant System for Mineral Exploration," in Expert Systems in the Micro-Electronic Age, D. Michie ed., Edinburgh Univ. Press, Edinburgh, 1979, pp. 153-167.

17. Engelmen, R, Terry, A., "Structure and Function of the CRYSALIS System", IJCAI 6, 1979, pp.250-256.

18. Erman, L.D., et al., "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty," Computing Surveys, Vol. 12, No. 2, June 1980, pp. 213-253.

19. Fagan, L.M. "VM: Representing Time-Dependent Relations in a Medical Setting," Ph.D. dissertation, Comp. Sci. Dept., Stanford Univ., Stanford, Ca., 1980.

20. Fahlman, S.E. NETL: A System for Representing and Using Real-World Knowledge, MIT Press, Cambridge, Mass., 1979.

21. Feigenbaum, E., Buchanan, G., and Lederberg, J. "Generality and Problem Solving: A Case Study Using the DENDRAL Program," Machine Intelligence 6, D. Meltzer and D. Michie, eds., Edinburgh University Press, 1971, pp. 165-190.

22. Forgy, C.L. The OPS5 User's Manual, Tech. Report Carnegie-Mellon University, 1980.

23. Gelernter, H.L., Sanders, A.F., Larsen, D.L., Agarwal, K.K., Boivie, R.H., Spritzer, G.A., and Searleman, J.E., "Empirical Explorations of SYNCHEM," Science v.197, 1977, pp.1041-1049.

24. Gevarter, William, B., "An Overview of Expert Systems", National Bureau of Standards Report, NBSIR 82-2505, 1982.

25. Haas, N., and Hendrix, G.G., "An Approach to Acquiring and Applying Knowledge", Proceedings of the First National Conference on AI, Stanford University, 1980, pp.235-239.

26. Hart, P.E., "Directions for AI in the Eighties," Sigart Newsletter, No. 79, Jan. 1982, pp. 11-16.

27. Hart, P.E., Duda, R.O., and Einaudi, M.T. A Computer-Based Consultation System for Mineral Exploration, Tech. Report, SRI International, Menlo Park, Calif., 1978.

28. Hayes-Roth, F., "AI The New Wave - A Technical Tutorial for R&D Management," (AIAA-81-0827), Santa Monica, CA: Rand Corp., 1981.

29. Hayes-Roth, F., Waterman, D.A., and Lenat, D.B. Building Expert Systems,Addison-Wesley, Reading, MA, 1983.

30. Jarké, M., Vassiliou, Y., "Coupling Expert Systems with Database Management Systems", Artificial Intelligence Applications for Business (W.Reitman, ed.), Ablex, to appear 1983.

31. King, J.J., "Query Optimization by Semantic Reasoning", Rep.no. CS-81-857, Computer Science Dept., Stanford University, 1981.

32. Kowalski, R.A., Logic for Problem Solving. North-Holland, New York, 1979.

33. Kunz, J.C., et al, "A Physiological Rule-based System for Interpreting Pulmonary Function Test Rules", Stanford Memo HPP-78-19, Stanford University, Computer Science Department, November, 1982.

34. Lenat, D.B. "AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search," Ph.D. dissertation Memo AIM-286, AI Laboratory, Stanford Univ., Stanford, Ca., 1976.

35. Lindsay, R.K., et al., Applications of Artificial Intelligence for Organic Chemistry: The DENDRAL Project, New York: McGraw-Hill, 1980.

36. Martin, N., et al., "Knowledge-Base Management for Experiment Planning in Molecular Genetics," Proc. Fifth Int'l Joint Conf. Artificial Intelligence, 1977, pp. 882-887.

37. Lowerre, B.T., "The HARPY Speech Recognition System," Ph.D. dissertation, Comp. Sci. Dept., Carnegie-Mellon Univ., Pittsburgh, Pa., 1976.

38. McDermott, D. "The PROLOG Phenomenon," Sigart Newsletter, No. 72, July 1980, pp. 16-20.

39. McDermott, D., "R1: A Rule Based Configurer of Computer Systems", Artificial Intelligence, 19(1), September, 1982.

40. McDermott, J., and Steele, B, "Extending a Knowledge-Based System to Deal with Ad Hoc Constraints," Proc. Seventh Int'l Joint Conf. Artificial Intelligence, 1981, pp. 824-828.

41. Michie, Donald, "Knowledge-based Systems," University of IL at Urbana-Champaign, Report 30-1601, June 1980.

42. Minsky, M. "A Framework for Representing Knowledge," The Psychology of Computer Vision, P.H. Winston, ed., McGraw-Hill, New York, 1975, pp. 211-277.

43. Moses, J. "Symbolic Integration: The Stormy Decade," Comm. ACM, Vol. 14, No. 8, 1971, pp. 548-560.

44. Mylopoulos, J. "An Overview of Knowledge Representation," Proc. Workshop Data Abstraction, Databases, and Conceptual Modeling, June 1980, pp. 5-12.

45. Mylopoulos, J., Bernstein, P., and Wong, H., "A Language Facility for Designing Database Intensive Applications", ACM, 5, 1980, pp.185-207.

46. Nau, D.S., "Expert Computer Systems", Computer, February, 1983, pp.63-85.

47. Nii, H.P., and Aiello, N., "AGE (Attempt to Generalize): A Knowledge-BasedProgram for Building Knowledge-Based Programs," Proceedings of the Sixth International Conf. on Artificial Intelligence, (IJCAI-79), Tokyo, Aug., 20-23, 1979, pp. 645-655.

48. Nilsson, N.J. Principles of Artificial Intelligencej, Tioga, Palo Alto, Calif., 1980.

49. Osborn, J., et al., "Managing the Data from Respiratory Measurements," Medical Instrumentation, Vol. 13, No. 6, Nov. 1979.

50. Pople, H., "Knowledge Based Expert Systems: The Buy or Build Decision", Artificial Intelligence Applications for Business (W.Reitman, ed.), Ablex, to appear 1983.

51. Quillian, M.R., "Semantic Memory", in Semantic Information Processing, M. Minsky (ed), 1968, pp.227-270.

52. Reggia, J., et al., "Towards an Intelligent Textbook of Neurology," Proc. Fourth Annual Symp. Computer Applications in Medical Care, 1980, pp. 190-199.

53. Robinson, J.A., A Machine Oriented Logic Based on the Resolution Principle, JACM, 1965, Vol.1, No.4, pp.23-41.

54. Sacerdoti, E.D., "Planning in a Hierarchy of Abstraction Spaces," Artificial Intelligence Vol. 5 No. 2, 1974, pp. 115-135.

55. Sacerdoti, E.D., A Structure for Plans and Behavior, American Elsevier, New York, 1977.

56. Schank, R.C. "Conceptual Dependency: A Theory of Natural Language Understanding." Cognitive Psychology, Vol. 3, No. 4, 1972.

57. Schank, R.C. Conceptual Information Processing, North-Holland, New York, 1975.

58. Schubert, L.K. "Extending the Expressive Power of Semantic Nets," Artificial Intelligence, Vol. 7, No. 2, 1976, pp. 163-198.

59. Shortliffe, Computer Based Medical Consultation: MYCIN, New York, American Elsevier, 1976.

60. Stallman, R.M., and Sussman, G.J. "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis," Vol. 9, Artificial Intelligence, 1977, pp. 135-196.

61. Stefik, M., et al., "The Organization of Expert Systems: A Prescriptive Tutorial", XEROX, Palo Alto Research Centers, VLSIi-82-1, January 1982.

62. van Emden, M.H., and Kowalski, R.A., "The Semantics of Predicate Logic as a Programming Language," J. ACM, Vol. 23, No. 4, 1976.

63. van Melle, W. "A Domain-Independent Production Rule System for Consultation Programs," Proc. Sixth Int'l Joint Conf. Artificial Intelligence, 1979.

64. Vassiliou, Y., Clifford, J., Jarke, M., "How does an Expert System Get Its Data?", NYU Working Paper CRIS#50, GBA 82-26 (CR), extended abstract in Proc. 9th VLDB Conf., Florence, October 1983.

65. Weiss, S.M., and Kulikowski, C.A. "EXPERT: A System for Developing Consultation Models," Proc. Sixth Int'l Joint Conf. Artificial Intelligence, 1979, pp. 942-947.

66. Weiss, S.M., et al., "A Model-Based Method for Computer-Aided Medical Decision-Making," Artificial Intelligence, Vol. 11, No. 2, 1978, pp. 145-172.

Knowledge Engineering and Fifth Generation Computers

Koichi Furukawa, Kazuhiro Fuchi
Institute for New Generation Computer Technology

1. Overview of the fifth generation computer project

Our fifth generation computer project has the primary objective of establishing knowledge information processing technology. In our view, knowledge technology technology consists of a wide range of technologies centering around knowledge engineering. For instance, it includes development of programming language and systems suited to knowledge engineering. Also included are man-machine communication technologies using natural languages, graphics and images.

The second objective of the project is to provide a drastic solution for the software problems. The productivity of software has been improved at a much slower rate than that of hardware, with the result that software accounts for a predominant proportion of the system cost. The project is aimed at solving this problem by entirely rebuilding computer science and technology upon new programming concepts such as logic programming and object oriented programming.

As an essential basis for achieving the two objectives, we chose a Prolog-based logic programming language [2]. This is because both knowledge information processing and program modularization require a powerful list processing capability and flexible data structure healing capability. As a language with these capabilities, we chose Prolog, skipping Lisp.

It is not easy to prove that one language is better than any other language, because there is no standard for measuring languages.

It is also difficult to convince people who believe that their favorite languages are the best. Nonetheless, we will try to give a technical justification for our choice of Prolog instead of Lisp.

Let us consider a short example of list processing, namely, appending two lists. The following is a Lisp program for the job.

```
(DEFUN APPEND (X Y)
   (COND (NULL X) Y)
      (T (CONS (CAR X)
         (APPEND (CDR X) Y)))))
```

where (DEFUN <name> <arg-list> <body>), defines a Lisp function with name <name>, arguments <arg-list>, and body <body>.

A corresponding Prolog program is given below:

```
append ( [ ] , Y, Y).
append ( [U,V] , Y, [U |W] ) : – append (V, Y, W.)
```

Let us give a brief explanation of the Prolog syntax. Character strings starting with a small letter such as append are constants and those starting with a capital letter such as "U", "V" are variable. [|V] is a list whose head is U and the rest is V. [ ] represents an empty list. "α: $-\beta$." is a sentence which means declaratively, "if β then α", and procedurally: "to achieve α, solve β". "α." is the abbreviation of "α : - true." and means that "α always holds".

Let us compare Prolog append with Lisp APPEND. The correspondence is shown below.

Lisp APPEND	Prolog append
COND	Two sentences
X	[U |V]
(CAR X)	U
(CDR X)	V
(APPEND (CDR X) Y)	W
(CONS (CAR X) (APPEND (CDR X) Y))	[U |W]

That is, a conditional sentence in Lisp corresponds to mutlipe sentences in Prolog in each case. In Prolog operations on list structures are visually represented by patterns. Note that the same form [$\alpha|\beta$] is used for cutting out elements from a list, namely as U = (CAR X), V = (CDR X), in the case of [U |V] and used for list construction, the same as (CONS U W), in the case of [U |W]. As can be seen from this example, Prolog simplifies list processing operations to a great extent. It is said that it took three months to discover this program after Prolog was developed. At that point in time Prolog started to evolve as a general-purpose list processing language.

Whatever its descriptive benefits may be, a programming language has little value if its execution speed is slow. A Prolog processing system was developed in Marseilles, France in 1971 [3], but it was useless because of a very slow execution speed and frequent memory overflow. It was with the advent of DEC 10 Prolog, developed by David Warren et al. [4], that this language began to attract attention. A Prolog interpreter and compiler were prepared from 1976 to 1977. The interpreter achieved a performance improvement of one order of magnitude over the Marseilles version and the compiler improved in performance by another order of magnitude. The execution speed of the compiler matched that of Lisp. This background may be considered a reason for our choice of Prolog. But there are other, and more important, reasons.

First, Prolog is superior in list processing description as illustrated in the example of append. Of particular importance is that it is possible to use undetermined variables as W in append ([U |V] , Y, [U |W]): - append (V, Y, W). This feature permits describing infinite structures and provides the basis for implementing Concurrent Prolog, which has parallel programming capability [5]. Functional programming languages such as Lisp achieves this ability based on the concept of delay evaluation but lack the ability to describe infinite structures.

Second, Prolog has an automatic backtracking mechanism, a powerful control mechanism for trial and error. Though something like a double-edged sword, this is particularly suited for describing problems calling for a search for solutions in the field of artificial intelligence.

Third, Prolog does not require any special gadgetry to achieve database capability. Lisp achieves database capability by using property lists attached to atoms. Prolog requires no such contrivance. It can create a database (of course, a small-scale database in main memory) in the same form as other Prolog sentences.

There are several other reasons. Prolog offer broad opportunities for sophisticated treatment of programs including simplification of program debugging [6] and synthesis, increased flexibility of module integration in modular programming [7], [8], and ease of optimization and partial execution of programs [9], [10]. These will help to solve various software problems – the second objecdtive of the project.

In addition to the foregoing technical reasons, attention should also be paid to how rapidly research and development efforts revolving around Prolog or logic programming languages are advancing and expanding in scale.

The fifth generation computer project is aimed at developing a new programming language as an extended version of Prolog and developing a computer dedicated to that language. And we hope this computer will be the prototype of general-purpose computers of the 1990s. The language will be highly flexible, and we think only the flexible is worthy to be called "general-purpose."

To give a total picture of the fifth-generation computer project, it is in order to touch on the hardware aspect of the project. It incorporates highly parallel computer architecture and VLSI technology. The highly parallel computer architecture is one of the few promising ways available to achieve a quantum jump in performance, but so far it has been thought of as hardly fit for general-purpose computers mainly because of its low compatibility with programming languages. One of the reasons for our selection of a logic programming language for the project is that we thought it would solve this problem. In the words of R. Kowalski, the logic programming langjage is the missing link between knowledge information processing and highly parallel computer architecture. Overall relationships of these are as shown in Figure 1.

2. The role of the fifth generation computer in knowledge engineering

Knowledge engineering is viewed as having great potential for growth, but is not yet past the stage of an experimental laboratory system. Feigenbaum pointed out that to make it an industry-level system, it is mandatory to enhance its software quality and to develop dedicated hardware with a sufficient processing speed, or about 100 times the speed of existing Lisp machines.

We believe our projected fifth generation computer can be an excellent tool that meets these requirements. For one thing, a highly parallel inference computer if developed will sufficiently meet at least the processing speed requirement.

Since it takes a considerable time to develop a full-scale highly parallel inference computer, we have to fill the gap somehow. Among the developments planned for the first three years of the project is a sequential Prolog machine. The performance of this machine will be about two orders of magnitude lower than that of the parallel inference computer and roughly equal to that of Prolog Compiler on PEC 2060.

Anyway, we can reasonably expect that even within the existing framework of technology, a performance improvement of at least one order of magnitude will be brought by the advance of device technology in a few years. The question is how to meet the other requirement of improving the software quality. This is not as simple a proposition as it may look. It has been the biggest challenge to software engineering and remains yet to be fundamentally solved even there.

Incidentally, as stated earlier, one of the major objectives of the fifth-generation computer project is to provide a drastic solution for software problems; and to achieve this objective, we chose a Prolog-based logic programming language to form the core of the project. The fact so far revealed by research in knowledge engineering is that the production system can be used as a software tool and that the future progress of knowledge engineering will require research and development work on the approach of making a model of a real world after the style of simulation languages and distributing problem solving capability to each system component, and programming systems for that purpose. Furthermore, it is important that such systems should be developed in a conceptually well-organized form to ensure an adequate level of programming efficiency.

Another question is whether these technological requirements can be met by logic programming languages. The prospects are encouraging, judging from the numerous experiments so far conducted with Prolog. Mizoguchi et al [11], developed expert systems based on Lisp and Prolog, respectively, and compared the systems. According to their findings, Prolog required about one fifth of the program coding volume required by Lisp, which means Prolog achieved a software productivity improvement of about one order of magnitude over Lisp since program development time is roughly proportional to coding volume. This fact has more implications than that. One is that ideas can be demonstrated immediately. So far, it has often been the case that when there was a good idea, the tremendous difficulty involved in its programming brought the progress of research on it to a standstill. But this problem is eased by the reduction in program development time.

The reason why Prolog allows such efficient programming is, in a nutshell, that it matches the human thought process better than other languages. Some people may think that a programming language is something like a mass of special codes. But in fact, it is a sort of full-fledged language. If one learns a programming language sufficiently, one becomes able to think in that language. And this is true of an assembler language, Fortran, Lisp or Prolog. The only difference among these programming languages is that each of them has a different forte as a tool for thinking. The remoter from the computer and the more abstract a programming language is, the better it matches the human thought process. In this sense we might say Prolog is closer to human language than any other programming languages.

The remaining major challenge is how to implement distributed problem solving with a logic programming language. Distributed problem solving depends heavily on a programming technique called object-oriented programming. With this technique, thinking centers around "objects" on computation and programs are regarded as describing behaviors of the objects. In the context of distributed problem solving, each component of a system to solve problems is an "object" and each associated problem solving machine is considered to simulate the behaviors of the object. In a computation method like this, the capability of communication between objedts plays a very important role, just as very close communication is needed when people work together.

The authors' group and others have recently been conducting active research on object-oriented programming in a logic programming language, and are working out a basic methodology [12], [13].

3. Concluding remarks

Knowledge information processing technology has great potential for opening up new computer applications. And the fifth-generation computer project is aimed at making this a reality as one of its objectives. It is one of several possible approaches, but we believe it offers the greatest possibilities and is of the best quality, technologically.

The greatest challenge to us in pursuing this approach in the years ahead is to disseminate this new culture. At the starting point of this movement, we need to implement a high-quality Prolog processing system on an existing commercial computer and make it available to everyone. Already, some Prolog processing systems have been developed, but their use, though gradually increasing, is not yet in full swing. It is also expected that a rather powerful sequential Prolog machine will appear as an intermediate result of the Fifth Generation Computer Project in Japan.

As for knowledge engineering per se, there will be a growing need for cooperation between computer scientists and other experts in the fields concerned. That is, an inter-disciplinary approach will be a must.

Reference

[1] Furukawa, K. "An Outlook for Fifth Generation Computer Software Technology," Electronic Communication Society Journal, Vol. 66, No. 4, special issue with articles on "Present Status and Future of Software Production Technology," 1983 (in Japanese).

[2] Yokoi and Fuchi, "Prolog, a Predicate Logic Language with Built-in Inference Mechanism," Nikkei Electronics, special issue (no. 300) with articles on "Artificial Intelligence", September 27, 1982 (in Japanese).

[3] Colmerauer, A., Kanoui, H., Pasero, R., and Roussel, P." Un Systeme de Comunication Homme-machine en Francais," Rapport, Groupe Intelligence Artificielle, Universite d'Aix Marseille, Luminy, 1973.

[4] Warren, D.H.D., Pereira, L.M., and Pereira, F. "PROLOG- The Language and its Implementation Compared with LISP," Proc. Symp. on AI and Programming Languages, SIGPLAN Notices, Vol. 12, No. 8, and SIGART Newsletters No. 64, August 1977, pp. 109–115.

[5] Shapiro, E.Y. "A Subset of Concurrent Prolog and Its Interpreter" Technical Report TR-003, ICOT, January, 1983.

[6] Shapiro, E.Y. "Algorithmic Program Debugging," Research Report 237, Yale University, Dept. of Computer Science, May, 1982.

[7] Farkas. Z. et al. "LDM: A Program Specification Support System". In Proc. First Int. Logic Programming conference. Marseille. 1982.
[8] Dogen: [K. Furukawa. R. Nakajima. A. Yonezawa] "Modularization and Abstraction in Logic Programming." ICOT Technical Report No. TR-022. 1983.
[9] Kahn. K. "A Partial Evaluator of Lisp written in Prolog" Proceedings of the First Logic Programming Conference. Marseille France. September 1982.
[10] H. Tamaki and T. Sato. A Transformation System for Logic Programs which Preserves Equivalence. ICOT Technical Report No. TR-018. July 1983.
[11] Mizoguchi. F., Miwa. K. and Honma. Y. "An Approach to Prolog Basis Expert System". Proc. of the Logic Programming Conference '83. Tokyo. March 22–24. ICOT. 1983.
[12] Shapiro. E.Y., Takeuchi. A. "Object Oriented Programming in Concurrent Prolog," New Generation Computing Vol. 1. No. 1. 1983.
[13] Furukawa. K., Takeuchi. A. and Kunifuji. S. "Mandala: A Knowledge Programming Language on Concurrent Prolog," ICOT TM-0028. 1983.

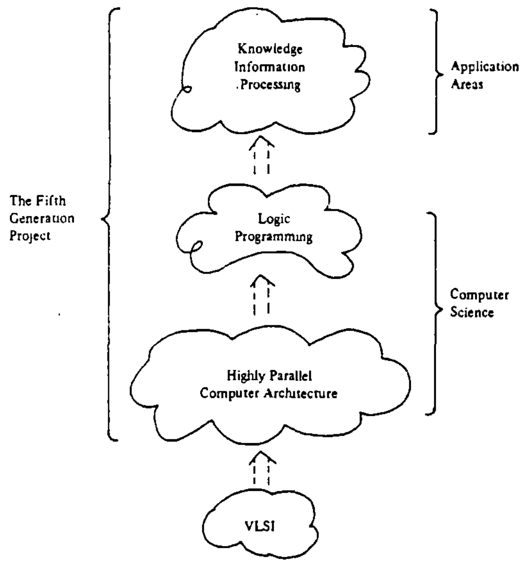

Figure 1

Logic Programming and Relational Databases

Kamran Parsaye

Silogic Inc.
6420 Wilshire Blvd, Suite 2000,
Los Angeles, CA 90048.

Introduction

Two significant developments in computer science in the past decade have been Relational Databases and Logic Programming. Relational databases were invented in the U.S. in the early 1970s, just as logic programming in Prolog was independently evolving in Europe. This paper discusses the language Prolog and its relation to relational databases.

Prolog

Prolog is a simple yet powerful language invented at the University of Marseille in the early 1970s. It is a declarative language whose design <u>includes</u> a relational database. Prolog computes relations, just as Lisp compute functions.

The origins of Prolog go back to Robert Kowlaski's ideas on programming in logic and to the work of Alain Colmerauer on natural language understanding. Colmeraire had been working on natural language grammers in Marseille during the 1960s, while Kowalski worked on logic programming in London. Following Kowalski's visit to Marseille in 1970, Colmeraure defined Prolog.

Prolog is a symbolic language based on pattern matching and IF-THEN rules. A Prolog program is simply a collection of these rules. Prolog uses an internal relational database to store such rules, and then applies them by pattern matching.

Prolog treats programs and data uniformly and thus combines programming and querying very effectively. It can thus be used as a relational query language as well as for general purpose programming.

A Brief Review of Prolog

The basic building blocks of Prolog programs are clauses. A clause in Prolog is a predicate name, called a functor, with some arguments. For instance

```
father(john,mary).
square(3, 9).
```

are clauses, where 'father' and 'square' are functors and 'john', and 'mary', 3 and 9 are arguments.

Arguments may be constants or variables, and conventionally, non-numeric constants are denoted by lower case letters, while variables must start with uppercase letters, e.g. as in father(X,mary).

In Prolog clauses can be asserted to be true, in which case they are included in the Prolog "database". The Prolog database contains all facts which are asserted to be true. For instance,

```
assert(father(john, mary)).
```

will include father(john,mary) in the database and

```
retract(father(john, mary)).
```

will remove it.

In Prolog, clauses are used to make sentences. A sentence in Prolog may be a simple unit clause, such as father(john, mary), or it may involve the conditional construct denoted by ":-", and better understood as "if".

For example, the conditional sentence

```
parent(X, Y) :- mother(X,Y).
```

means that "for all X and Y", parent (X, Y) is true if mother(X, Y) is true. Thus essentially "A :- B" means that A is logically implied by B.

Clauses on the right hand side of a ":-" can be joined together by "and" and "or" constructs denoted by "," and ";" respectively, as in

```
parent(X, Y) :- mother(X,Y); father(X,Y).
```

which means: "for all X and Y", parent(X, Y) is true if either mother(X, Y) is true "or" father (X, Y) is true.

Let us make two simple technical notes here. First that sentences in Prolog must end with a period. Second that due to the universal quantification above, the range of each variable in Prolog is essentially a sentence, i.e. two occurances of the same variable name within two sentences are totally unrelated. The Prolog compiler will internally rename variables to avoid conflicts.

Variables on the right hand side of a conditional which do not appear on the left hand side are existentially quantified. For instance the sentence

```
grandfather(X, Y) :- father(X, Z), parent(Z, Y).
```

means that "for all X and Y", X is the grandfather of Y if "there exists" some Z in the database, such that X is the father of Z and Z is a parent of Y.

In Prolog, conditional clauses may be stored in the database just as data are, i.e. programs are really treated as data in a database. This uniform view of both programs and data as items in a high level database is perhaps the major reason for the elegance of the Prolog programming style.

Once one has adopted this database view of programming, one may naturally wonder about queries to the database. Simple queries may relate to simple facts such as: "Is father(john, mary) true in the database?", which may simply require a look up in the database. However, one may also ask more complex queries.

We generally refer to an attempt to answer a query in the Prolog database as an attempt to satisfy a goal (or to prove a goal). For instance, in the example above "father(john, mary)" is the goal, and it can be satisfactorily proved if father(john, mary) has been asserted in the database.

One may also try to prove goals with variables, in which case Prolog will try its best to find a match for the variables to satisfy the goal. For instance, an attempt to prove "father(X, mary)" will succeed provided that the condition (X = john) is true. Note that this is not an assignement (Prolog is assignment free), but a binding of a variable to a value as in pure Lisp. Such bindings are discarded upon the completion of the query.

Now, how about conditional clauses? Since the interpretation of the conditional construct ":-" is that the right hand side logically implies the left hand side, the validity of the left hand side can be established by proving the right hand side. This new goal may itself in turn be part of a conditional clause,..., and so on. Thus execution of programs in Prolog essentially consists of attempts to establish the validity of goals, by chains of pattern matching on asserted clauses.

To prove a goal Prolog searches its database for a clause that would match the goal, by using the process of unification [Robinson 65]. If a conditional clause whose left hand side matches the goal is found, Prolog tries to satisfy the set of goals on the right hand side of ":-" in a left to right order. If no matching clause can be found, failure will be reported.

It must be noted that Prolog includes no explicit negation symbol, and negation is essentially treated as unprovability, i.e. the failure to establish a goal from a set of axioms [Clark

79]. This closely resembles the closed world assumption [Reiter 78].

If Prolog does not succeed in establishing a goal in a chain of deductive goals at a first try, it will backtrack, i.e. go to the last goal it had proved and try to satisfy it a different way. For instance, suppose that we have the sentences (or program)

```
parent(X, Y) :- mother(X,Y); father(X,Y).               (*)
grandfather(X, Y) :- parent(Z, Y), father(X, Z).      (**)
```

and that the following facts have also been asserted:

```
father(john, mary).
father(paul, john).
mother(jennifer, mary).
```

Then to prove "grandfather(X, mary)", by using (*) and (**) above, first the goal "parent(Z, mary)" will be tried. This in turn will result in an attempt to prove "mother(Z, mary)" and will succeed with (Z = jennifer). Then, going back to the "grandfather" clause again, the next goal in the conjuntion should be proved. So "father(jennifer, Y)" will be tried and will fail. At this point Prolog will go back (i.e. will backtrack), discard the assumption (Z = jennifer) and try to prove "parent(Z, Y)" again. This time (Z = john) will result, after trying "father(Z, mary)". The eventual binding (Y = paul) will be returned after trying "father(X, john)".

Let us note that in the grandfather "program" here there are no explicit input or output parameters, i.e. one may either invoke grandfather (X, mary), or grandfather(paul, Y). This style of declarative programming in Prolog can often be used to great advantage to develop software very rapidly. Prolog programs can often be run both forwards and backwards.

On one hand, the series of steps taken by the Prolog compiler in proving a goal essentially amount to deduction. On the other hand an attempt to prove a goal "father(X, Y)" can also be looked at as a procedure call to the predicate father. Thus the use of the term "logic programming" is quite apt here.

Calls in Prolog can also be recursive, as in

```
connected(X, Y) :- edge(X, Z), connected(Z, Y).
```

which deals with connectivity in graphs described in terms of edges. The Prolog compiler [Warren 77] uses tail recursion optimization to great advantage in such cases.

Now, for expression evaluation. In the author's opinion, one of the most inconvenient features of symbolic languages such as Lisp has been the relation between quotation and evaluation. The

Prolog approach to evaluation is exactly the opposite of Lisp, i.e. evaluation does not take place until it is forced to. This is specially relevant to arithmetic expressions and removes the need for quotes. Thus (2 + 3) can be evaluated to 5 when the need arises, by using the Prolog infix operator "is", i.e. "X is (2 + 3)" binds X to 5. However again note that this is not assignement.

Many more examples of Prolog programs, and a more detailed description of the language and its use may be found in [Clocksin & Mellish 81].

Relations and Logic

Relational databases are based on relations. Prolog is based on predicate logic. Expressing the semantics of the relational model in terms of predicate calculus [Gallier & Minker 78] can be used to demonstrate the essential connection between Prolog and relational databases.

This connection can also be seen with simple examples by viewing each relation as a predicate which asserts the truth of facts. For example consider the relation "father" which includes the tuple <john, mary>. In Prolog one can assert the truth of an instance of the "father" predicate. The Prolog equivalent of the tuple from the "father" relation is thus father(john, mary).

The correspondence between "data" in relational databases and Prolog assertions is thus quite evident. However, this correspondence can even be extended to the semantic level. The fact that both Prolog and relational databases are subsets of first order predicate calculus suggests a resemblence between logic programming and the relational model, but in itself is inadequate for stating equivalence. The reason for equivalence lies in the characterization of the semantics of relations in terms of relational dependencies.

Logic and Dependencies

The semantics of the relational model is often expressed in terms of relational dependencies [Gallier & Minker 78]. The first dependencies to be discovered were the functional dependencies invented by Codd in 1972, and later axiomatized by Armstrong [Armstrong 74]. Later, Zaniolo and Fagin independently invented multivalued dependencies for decomposition [Zaniolo 76] [Fagin 76].

The years 1976 to 1980 saw the invention of many other dependencies [Paredeans 78], [Sagiv & Walecka 79], [Parker & Parsaye 80], [Sadri & Ullman 80], etc. This was because no group of dependencies was found which could adequately describe the semantics of the relational model.

In the fall of 1979, Ronald Fagin discovered an interesting set of dependencies which captured all of the previously postulated dependencies and which very naturally expressed the semantic of the relational model. He called these the "implicational dependencies" [Fagin 80].

Fagin noticed that implicational dependencies were a subset of first order logic whose properties had previously been studied by Alfred Horn [Horn 51]. This subset of first order logic has been known to logicians as Horn clauses since the 1950s*.

What Fagin did not realize at the time was that implicational dependencies, i.e. Horn Clauses, are _exactly_ the sentences used by the language Prolog and can thus be executed as a programming language. Similarly, the logic programming community was neither involved nor interested in relational dependencies until the early 1980s.

Finally, with the avilability of the DEC-10 Prolog compiler in the U.S. and the growing database interests in Europe a merging of the views on logic programming and relational databases began in the early 1980s [Dahl 82], [Parsaye 83]. Today the interest in logic programming and relational databases is growing very rapidly.

Implementations

Based on the previous semantic evidence, it seems plausible to assume that one can add a relational database system to Prolog with great ease. In the spring of 1983, in association with D.S. Parker and P.E. Eggert, we undertook the task of putting these ideas into a commercial database system called Maxwell at Silogic, Inc.

Maxwell includes full features of the relational model but manages the storage of not only "data" but "knowledge", i.e. rules of inference on secondary strage devices. It is thus the first "knowledge base" management system to be implemented.

In implementing Maxwel, we became aware of many subtle implementation issues which needed very careful treatment. As one might have guessed, these issues were not obvious at the semnatic level! The implementation was first prototyped in Prolog itself and then rewritten in C, under Unix. Details of this implementation effort may be found in [Eggert, Parker & Parsaye

* As a very technical aside, let us note that the basic property of Horn clauses which makes them useful and interesting is that they are the maximal subset of first order logic which admits "typical models" or "functionally free algebras" [Tarski 36]. In the context of the relational model, Fagin called these "Armstrong relations". In higher order logic, [Parsaye 81] discusses "relatively free algebras".

84].

Our experience showed that adding databases to Prolog is indeed possible, but involves many delicate implementation subtleties which if not observed can degrade performance. The need for the careful selection of algorithms in such system can not be over-emphesized.

Some Future Directions

Relational databases and Prolog are moving fast towards a common destination. This holds true both in terms of functionality and performance.

Processing deductive natural language queries in Prolog is now a well-tested practicality [Warren & Pereira 82]. Similarly, many relational databases are now adapting natural language query systems. Today Prolog compilers are achieving impressive execution speeds even on medium and small sized machines.

One of the main advantages of Prolog's uniform approach to data and knowledge is in the development of expert and knowledge based systems, particularly in applications where the expert system needs to access a database, e.g. as in [Blum 82].

With the selection of Prolog as the language at the core of Japan's Fifth Generation Computing Systems project [FGCS 81], powerful computing systems combining logic programming and relational databases are likely to be forthcoming in the near future.

Combining the deductive powers and the ease of use of Prolog with the data management capabilities of large relational databases results in powerful computing environments [Ohsuga 82]. Such environments will dominate computing in the 1980s.

References

[Armstrong 74]

"Dependency Structures of Database Relationships, Proceedings of the 1974 IFIP Congress, Amsterdam.

[Blum 82]

"The RX Project", Computer Science Department, Stanford University.

[Buchanan 81]

"Research on Expert Systems", Heuristic Programming Project Report, Stanford University.

[Clark 78]

"Negation as failure", in [Galler & Minker 78].

[Clark 80]

"Prolog, A language for Implementing Expert Systems", in 'Machine Intellingence 10', edited by P. Hayes and D. Michie.

[Clocksin & Mellish 81]

"Programming in Prolog", Springer Verlag.

[Colmerauer 75]

"Les Grammaires de Metamorphose", Groupe d'Intelligence Artificialle, Marsille-Luminy, 1975, reprinted in Lecture Notes in Computer Science Vol 63.

[Dahl 82]

"On Database Systems Development through Logic", ACM Transactions on Database Systems, Vol 7. No 1.

[Deliyanni & Kowalski 79]

"Logic and Semantic Networks", Communications of the ACM, Vol 22. No 3.

[Eggert, Parker & Parsaye 84]

"Prototyping in Prolog: Experiences and Lessons", IEEE Conference on Logic Programming, Atlantic City, February 1984.

[Fagin 76]

"A New Normal Form for Relational Schemas", Research Report, IBM San Jose Research Laboratory.

[Fagin 80]

"Horn Clauses and Database Dependencies", Proceedings of the 1980 ACM Conference on Foundation of Computer Science.

[FGCS 81]

"Proceedings of the Fifth Generation Computer Systems Conference", Tokyo, Japan.

[Gallier & Minker 78]

"Logic and Databases", Plenum Press, New York, 1978.

[Horn 51]

"On Sentences which are true of Direct Unions of Algebras", Journal of Symbolic Logic, Vol 16.

[Kowalski 79]

"Logic for Problem Solving", North Holland.

[Kowalski 81]

"Logic as a Database Language", Department of Computing, Imperial College London.

[Lloyd 81]

"Implementing Clause Indexing in Deductive Database Systems", Technical Report, Department of Computer Science, University of Melbourne.

[Michie 80]

"Expert Systems in the Micro-electronic Age", Edinburgh University Press.

[Nicolas 78]

"Logic and Databases", in [Gallier & Minker 78].

[Ohsuga 82]

"Knowledge-Based Systems as a New Interactive Computer System of the Next Generation", in Kitagawa ed.: "Japan's Annual Reviews in Computer Science and Technologies", North Holland.

[Paredeans 79]

"Transitive Dependencies in a Databse System", Technical Report, MBLE, 1979.

[Parker & Parsaye 80]

"Interferences Involving Embedded Multi-Valued Dependencies and Transitive Dependencies", Proceedings of the 1980 ACM SIGMOD Conference.

[Parsaye 81]

"Higher Order Abstract Data Types", Ph.D. Dissertetion, Computer Science Department, UCLA.

[Parsaye 83]

"Database Management, Knowledge Base Management and Expert System Development in Prolog", Proceedings of the ACM SIGMOD Database Week Conference, San Jose, May 1983.

[Reiter 78]

"On Closed world Databases", In [Gallier & Minker 78].

[Robinson 65]

"A Machine oriented Logic Based on the Resolution Principle", Journal of the ACM, Vol 12 No.1.

[Robinson 80]

"Programming by Assertion and Query", in [Michie 80].

[Sadri & Ullman 80]

"A Large Class of Dependencies for Relational Databases", Proceedings of the 1980 ACM Conference on Foundation of Computer Science.

[Sagiv & Walecka 79]

"Subset Dependencies and EMVDs", Technical Report, Depatment of Computer Science, University of Illinois.

[Warren 81]

"Efficient Processing of Interactive Relational Database Queries Expressed in Logic", Proceedings of the 1981 VLDB.

[Warren, Pereira & Pereira 77]

"Prolog, The Language and its Implementation Compared with Lisp", Proceedings of the ACM Symposium on AI and Programming Languaguages.

[Warren & Pereira 81]

"The CHAT-80 System", Technical Report, Department of Computer Science, University of Edinburgh.

[Zaniolo 76]

"Analysis and Design of Schemata for Relational Databases", Ph.D. Dissertation, Department of Computer Science, UCLA.

The Knowledge Engineering Process

John McDermott

Department of Computer Science

Carnegie-Mellon University

Pittsburgh, PA 15213[1]

1. Four stages in the life of a knowledge engineer

During the past fifteen years, Artificial Intelligence researchers have explored techniques for bringing large amounts of domain knowledge to bear in solving ill-structured problems. The process of building a knowledge-based system (or expert system as they have come to be called) ordinarily consists of the following four stages:

- Task definition
- Initial design
- Earnest knowledge extraction
- The aftermath, technology transfer

If a knowledge-intensive problem has been identified, the first stage in building a system to assist in solving the problem is to delimit its task; since solving knowledge-intensive problems is often a communal endeavor and since the problems are ordinarily not cleanly bounded, there is frequently a real issue of determining just what role the expert system is to play. Once the task has been defined, the developer must create an initial design for the system on the basis of the expected characteristics of the as yet uncollected domain knowledge; the trick here is to figure out how that knowledge can best be exploited to guide the system's problem solving behavior. Once there is a design, the knowledge extraction process will have a focus; thus the skeletal system implicit in the design can begin to be fleshed out with knowledge. At some point, long before the system has become expert in its task, it will be able to provide useful assistance; at that point issues arise of how to continue to develop the system in an operational environment.

1.1. Task definition

In order to appropriately delimit a task, it is necessary to define the role the expert system will play relative to others who perform complementary tasks. If there is a large amount of knowledge that is relevant to some problem, then it is often the case that the required expertise is distributed among several people. Each individual's role is defined both by his or her area of competance and then by a level of competance within that area. When an expert system is developed to assist with some problem, the first question that has to answered is whether the system will assume an existing role or

[1]This paper borrows heavily from the material in [McDermott 80] and [McDermott 83].

carve out some new role. How that question is answered will have a strong impact both on how knowledge acquisition is handled and on how transfer to an operational environment is carried out.

1.2. Initial design

The stage in building an expert system that is least well understood is that of creating an initial design. What sets an expert system apart from other computer programs is that it is able to perform its task because it can bring large amounts of domain knowledge to bear at each step; this knowledge gives the expert system its problem solving power. But in order for an expert system to be able to exploit its knowledge, the knowledge must be represented in such a way that all of it that is relevant at any step is immediately available to the system. The principal problem in developing an expert system is in determining how to represent the knowledge so that it is always available when needed. In order to extract knowledge from experts, it is first necessary to understand how that knowledge is going to be represented; but without already having that knowledge, it is not clear what has to be represented. At present, all we can do in this circumstance is fall back on general arguments for representational adequacy. In the future, we will hopefully have enough of an understanding of the representational demands of a wide variety of task types that the particular representational demands of some new task will be clear on the basis of the similarities of that task to one or more of those task types.

1.3. Knowledge extraction

All of the expert systems that have been developed to date have acquired their knowledge over a long period of time. In the typical case, a relatively small but central fraction of the domain knowledge has been collected by a knowledge engineer interacting over a period of a few weeks or months with a domain expert. This knowledge has been used to build a novice system that can behave appropriately in commonly occurring situations, but that is almost total insensitive to special circumstances. Knowledge is then extracted from an expert by asking the expert to observe the behavior of the novice, point out mistakes the novice makes, and for every mistake, indicate what knowledge the novice is lacking or has wrong. Over a period of months or years, this process uncovers enough of the domain knowledge so that the novice grows to be an expert. Variations in this process occur depending on the source of the expertise. If a system is to assume an existing role (ie, if there are human experts who perform the same task), then the source of the knowledge is those who play that role. If the system is to carve out a new role, part of the knowledge acquisition problem is identifying the sources of knowledge.

1.4. Technology transfer

A human expert does not stop acquiring knowledge of his domain at some arbitrary point in his life; there is always room for him to become more expert. Likewise, the development of an expert system is never finished. Thus expert systems have no need for maintainers, but they do always need developers even after they have moved to an operational environment. This need poses a problem since the current demand for people with some background in artificial intelligence far exceeds the supply. If expert systems are to have an appreciable impact on industry, ways must be found to provide a significant number of people with an understanding of AI techniques.

Since 1978, Carnegie-Mellon University and Digital Equipment Corporation have been collaborating on the development of a number of expert systems. R1, the first of these systems to be developed, is a computer system configurer [McDermott 82, McDermott 81]. Since R1 is now almost five years old and has been used for much of that time by Digital's manufacturing organization, it can serve as a good example of the four stages. The next section of the paper describes R1's task. Then in the third section R1's progress through the four stages is discussed. For discussions of other expert systems which may help to broaden the picture presented here, see [Feigenbaum 77] and [Stefik 82].

2. Computer System Configuration

2.1. The task domain

Digital differs from most computer manufacturers in the degree of flexibility it allows its customers in component selection; rather than marketing a number of standard systems, each with a limited number of options, Digital markets processors with relatively large numbers of options. One of the results of this marketing strategy is that many of the systems it sells are one of a kind, and consequently each poses a distinct configuration problem. A computer system configurer has two responsibilities: he must insure that the system is complete (ie, that all of the components required for a functional system are present), and he must determine what the spatial relationships among the components should be. Because a typical system has 100 or so components which have a large number of different possible interrelationships, a significant amount of knowledge is required in order to perform the configuration task competently.

Before the advent of R1, the configuration task was performed by people of varying degrees of expertise at various points in the order processing pipeline. Each order was examined at least twice by people known as technical editors. The first examination was to insure that the order was complete (ie. that the particular set of components on the order were sufficient for a functional system). The second examination, though concerned with completeness, also had the purpose of specifying what the relationships among the components should be. Because the technical editor's task is fairly tedious, it is not a position that engineers aspire to; thus the people who have this job tend to have fairly weak engineering backgrounds. A technical editor can, however, call upon an engineer who understands computer system issues whenever he encounters a problem that is beyond his competence.

2.2. R1's capabilities

R1 is implemented in OPS5 [Forgy 81]. OPS5 is a general-purpose rule-based language; like other rule-based languages, OPS5 provides a rule memory, a global working memory, and an interpreter which tests the rules to determine which ones are satisfied by the descriptions in working memory. An OPS5 rule is an IF-THEN statement consisting of a set of patterns which can be matched by the descriptions in working memory and a set of actions which modify working memory when the rule is applied. On each cycle, the interpreter selects a satisfied rule and applies it. Since applying a rule results in changes to working memory, different subsets of rules are satisfied on successive cycles.

OPS5 does not impose any organization on rule memory; all rules are evaluated on every cycle.[2]

OPS5's two memories have been augmented, for this application, with a third. This memory, the product description data base, contains descriptions of over 4000 components. Each entry in the data base consists of the name of a component and a set of forty or so attribute/value pairs that indicate the properties of the component that are relevant for the configuration task. As R1 begins to configure an order, it retrieves the relevant component descriptions from this data base. As the configuration is being generated, working memory grows to contain descriptions of partial configurations, results of various computations, and context symbols that identify the current subtask.

Production memory contains all of R1's permanent knowledge about how to configure computer systems. R1 currently has 2800 rules that enable it to perform the task. These rules can be viewed as state transition operators. The conditional part of each rule describes features that a state must possess in order for the rule to be applied. The action part of the rule indicates what features of that state have to be modified or what features have to be added in order for a new state that is on a solution path to be generated. Each rule is a more or less autonomous piece of knowledge that watches for a state that it recognizes to be generated. Whenever that happens, it can effect a state transition. If all goes well, this new state will, in turn, be recognized by one or more rules; one of these rules will effect another state transition, and so on until the system is configured. English translations of two sample rules are shown in Figure 1.

It is usual, when discussing an artificial intelligence system, to distinguish the matching of forms and data from search; for example, in discussing the amount of search occurring in a resolution theorem prover, the unification of clauses is considered to be part of the elementary search step. But Match is also a method for doing search in a state space [Newell 69]; it is analogous to methods such as Hill Climbing or Means-ends Analysis, though much more powerful. The characteristic that distinguishes Match from other Heuristic Search methods is that in the case of Match the conditions (tests) associated with each state are sufficient to guarantee that if a state transition is permissible, then the new state will be on a solution path (if there is a solution path). Thus with Match, false paths are never generated, and so backtracking is never required. Match is well suited for the configuration task because, with a single exception, the knowledge that is available at each step is sufficient to distinguish between acceptable and unacceptable paths. The subtask that cannot always be done with Match alone is placing modules on the unibus in an acceptable sequence; to perform this subtask, R1 must occassionally generate several candidate sequences.

The fan-in and fan-out of R1's rules provide a measure of the degree of conditionality in the configuration task. The *fan-in* of a rule is the number of distinct rules that could fire immediately before that rule; the *fan-out* is the number of distinct rules that could fire immediately after the rule. The average fan-in and fan-out of R1's rules is 3; about a third of the rules have a fan-in of 1, and a third have a fan-out of 1. The graph of possible rule firing sequences, then, has about 2800 nodes (one for each rule); each of these nodes has, on the average, three edges coming into it and three

[2]OPS5's cycle time, though it is essentially independent of the size of both production memory and working memory [Forgy 82], depends on particular features of the production system (eg, the number and complexity of the conditions and actions in each production); the average cycle time for OPS5 interpreting R1 is about 50 milliseconds on a VAX-11/780.

```
ASSIGN-UB-MODULES-EXCEPT-THOSE-CONNECTING-TO-PANELS-4

     IF:        The current context is assigning devices to unibus modules
                And there is an unassigned dual port disk drive
                And the type of controller it requires is known
                And there are two such controllers
                           neither of which has any devices assigned to it
                And the number of devices that these controllers can support is known

   THEN:        Assign the disk drive to each of the controllers
                And note that the two controllers have been associated
                           and that each supports one device


PUT-UB-MODULE-6

     IF:        The current context is putting unibus modules in backplanes in some box
                And it has been determined which module to try to put in a backplane
                And that module is a multiplexer terminal interface
                And it has not been associated with any panel space
                And the type and number of backplane slots it requires is known
                And there are at least that many slots available
                           in a backplane of the appropriate type
                And the current unibus load on that backplane is known
                And the position of the backplane in the box is known

   THEN:        Enter the context of verifying panel space for a multiplexer
```

Figure 2-1: Two Sample Rules

going out. It should be clear that unless the selection of which edge to follow can be highly constrained, the cost (in nodes visited) of finding an adequate configuration (an appropriate path through the rules) will be enormous. It is in this context that the power of the Match method used by R1 becomes apparent. When R1 can configure a system without backtracking, it finds a single path that consists, on the average, of about 800 nodes. When R1 must backtrack, it visits an additional N nodes, where N is the product of the number of unsuccessful unibus module sequences it tries (which is rarely more than 2) and the number of nodes that must be expanded to generate a candidate unibus module configuration (which is rarely more than 300).

3. Building R1

3.1. Task definition

Several factors influenced the initial definition of R1's task. First, it was simply assumed that R1 would perform the task that the technical editor responsible for specifying the relationship among components performed. A system which simply checked for completeness would have been of use, but at that time, Digital's primary concern was with finding a way to insure consistency among configurations. Second, the possibility of developing a system that could reason like an engineer when encountering a previously unencountered problem was never considered; as a result, R1's knowledge turned out to be the surface knowledge of the technical editor, rather than the deeper knowledge of the engineer. Third, since a technical editor is ordinarily responsible for a single system type (eg, a PDP-11/44, a VAX-11/780), it was assumed that R1's initial task would be to configure just a single system type. But it was not at all clear which system type was the right one to start with. The alternatives were either one of the PDP-11 systems or the VAX-11/780. The main reason for favoring a PDP-11 was that configuration was more of a problem for the PDP-11 systems. The main reason for favoring the VAX-11/780 was that it was easier to configure, primarily because the number of supported components was, at that time, so much smaller. It was decided that the initial task should be VAX-11/780 configuration. As it turned out, this was a fortunate choice since it allowed R1 to become a good novice configurer more quickly and as a result gain increasing amounts of cooperation from those people who had knowledge it needed.

3.2. Initial design

After we had some understanding of what was involved in the configuration task at an abstract level, but before we had more than a few examples of the knowledge technical editors have in their heads, we made two basic design commitments which implicitly defined the structure that R1 would have. One of these was to encode the domain knowledge as situation/action rules; the other was to view the configuration task as a collection of loosely coupled subtasks. Each of R1's rules defines a situation in which some particular configuration step is required. Since a rule is applied only when the descriptions in working memory satisfy the condition part of that rule, R1's rules do not specify sequences of steps to be performed, but rather specify the possible steps without committing in advance either to particular steps or to the sequence in which the steps should be performed. R1 relies on this characteristic of rule-based systems in two ways. (1) Because there is so much variability in the steps required for configuring different systems, enumerating all possible sequences of steps would be an almost impossible task; encoding knowledge in rules permits each small piece of situation-specific knowledge to recognize when it is relevant and thus dynamically discover the appropriate next step. (2) Because the knowledge that R1 needs in order to become an expert configurer can only be acquired over time, new knowledge must be continuously assimilated; since a rule defines the type of situation in which it is relevant, rules do not interact very strongly and thus adding new rules is straightforward.

To say that the configuration task can be viewed as a collection of loosely coupled subtasks means that each piece of configuration knowledge can be associated with a particular subtask. When the nature of a task allows such partitioning of knowledge, issues of control are ordinarily easier to resolve. Because rule-based systems are data-driven, it can happen that rules which are satisfied but have no relevance to the subtask currently being performed are applied before (or even instead of) rules that bear on the current subtask. We decided to avoid this problem in R1 by including, in the condition part of each rule, a description of the particular subtask that the rule is relevant to. When a rule (relevant to a particular task) recognizes that some subtask needs to be performed before the current task can be completed, that rule asserts a description of the subtask which needs to be performed.

3.3. Knowledge extraction

After one week's interaction with a Digital engineer, it took about three months to develop an initial, primitive version of R1. This novice version had about 250 situation/action rules; this amount of knowledge enabled R1 to configure very simple VAX-11/780 systems without making any mistakes. Over a period of about five months, expert configurers observed R1's performance and indicated what knowledge R1 was lacking; this knowledge was encoded as rules and added to R1. A total of about one man-year was spent in getting R1 to the point where Digital could formally evaluate it. At 750 rules, R1's performance was fairly impressive; the experts concluded that it had almost all of the knowledge it needed, so it began to be used by Digital to configure all VAX-11/780 systems. R1 now has more than 2800 rules. Some of this additional knowledge was required to enable R1 to configure systems other than the VAX-11/780. But much of it is knowledge that was simply overlooked by the experts. The experts did not think to supply this knowledge until R1 tried to configure systems which could not be configured correctly without it. Because some of this knowledge is very rarely needed, it has taken several years to uncover some of it.

3.4. Technology transfer

As R1 was being developed at CMU, little attention was paid to the question of who at Digital was going to maintain and continue to develop R1 once it moved to Digital. R1 moved to Digital in January of 1980, and a small group of programmers was given the responsibility of seeing that R1's potential was realized. For about a year, the people at CMU who had developed R1 provided a significant amount of assistance. But by the end of 1980, the group at Digital had became sufficiently familiar with R1 and with the AI techniques it embodies to be able to proceed without additional assistance from CMU. During the past three years, the people at Digital have extended R1 from a 1000 rule system to a 2800 rule system.

The number of people who have worked directly on R1 has, over the past three years, remained at about ten. Three of these people are engineers who are responsible for identifying the configuration knowledge R1 lacks on the basis of mistakes it makes and for adding new product information to the data base. Four of the people are programmers; the engineers tell them what configuration knowledge R1 is lacking and they determine how to represent this knowledge in the form of rules. The other three people are R1's interface to the user community. Considering that none of these people had any background with AI when they began their jobs, they have accomplished a great deal.

References

[Feigenbaum 77] Feigenbaum, E. A.
The art of artificial intelligence.
In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 1014-1029. MIT, 1977.

[Forgy 81] Forgy, C. L.
The OPS5 user's manual.
Technical Report, Carnegie-Mellon University, Department of Computer Science, 1981.

[Forgy 82] Forgy, C. L.
Rete: A fast algorithm for the many pattern/many object pattern match problem.
Artificial Intelligence 19(1), 1982.

[McDermott 80] McDermott, J.
R1: an expert in the computer systems domain.
In *Proceedings of the First Annual National Conference on Artificial Intelligence.* Stanford University, 1980.

[McDermott 81] McDermott, J.
R1's formative years.
AI Magazine 2(2), 1981.

[McDermott 82] McDermott, J.
R1: A rule-based configurer of computer systems.
Artificial Intelligence 19(1), 1982.

[McDermott 83] McDermott, J.
Building expert systems.
In *Proceedings of the New York University Symposium on Artificial Intelligence Applications for Business.* New York University, 1983.

[Newell 69] Newell, A.
Heuristic programming: ill-structured problems.
In Aronofsky, J. S. (editor), *Progress in Operations Research*, pages 361-414. John Wiley and Sons, 1969.

[Stefik 82] Stefik, M., J. Aikins, R. Balzer, J. Benoit, L. Birnbaum, F. Hayes-Roth, and E. Sacerdoti.
The organization of expert systems, a tutorial.
Artificial Intelligence 2(18), 1982.

A Review of the Prospector Project

by
John E. Reiter
Syntelligence, Inc.*

I. Introduction

This paper summarizes the Prospector project, a ten-year effort at SRI International to develop techniques and tools to assist with decision making problems in mineral exploration and in other areas[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]. During that time, the project has shifted from being a pure research effort to being largely concerned with the transfer of the technology to various other organizations. It seems appropriate at this time to step back and review the project, not only to describe some of the important features of Prospector but also to summarize what has been learned.

II. The Design of Prospector

The Prospector system is a well-known example of a class of programs that have come to be known as *expert systems*. The phrase "expert system" has been applied to so many different programs that there is considerable confusion about its meaning. For us, an expert system is a computer program that uses explicitly represented knowledge and computational inference procedures to solve problems normally requiring significant human expertise.

The goal of Prospector is to play a role analogous to that of a human consultant who must deal with problems that can be characterized as diagnosis or classification problems; that is, given a set of (possibly uncertain) case-specific data, produce a set of possible explanations that summarize or interpret that data. Although Prospector has been primarily developed for geological applications, the system is for the most part domain independent.

A. Inference Networks

The *knowledge base* that Prospector uses to interpret a set of data largely consists of a set of *plausible inference rules* of the form:

IF E THEN (to some degree) H,

where E is a proposition that represents some piece of evidence and H is a proposition that denotes an hypothesis. E is sometimes called the *antecedent* of the rule and H the *consequent*. A particular piece of evidence E can be on the left hand side of any number of rules, and a particular hypothesis H can be on the right hand side of any number of rules. Also, rules can be chained together by allowing H to be on the right side of one rule but also on the left of one or more other rules (thus blurring the distinction between evidence and hypothesis).

One can view a collection of propositions and rules as an *inference network* by representing propositions as nodes and rules as arcs directed from evidence to hypothesis. The set of nodes with no outgoing arcs represent *top-level hypotheses*. The nodes with no incoming arcs represent *external evidence* for which information is potentially acquirable from some information source. The remaining nodes are called *intermediate hypotheses*. The problem, then, is to acquire evidence so that one can establish a top-level hypothesis.

*This paper was begun when the author was affiliated with SRI International. The author's current address is 800 Oak Grove Avenue, Suite 201, Menlo Park, CA 94025.

B. Methods for Handling Uncertain Information

In general, the truth or falsity of a top-level hypothesis cannot be established with complete certainty. Thus, Prospector associates a subjective probability value with every proposition to measure the degree to which it is believed to be true. As in Bayesian decision theory, it is assumed that in the absence of evidence any hypothesis H has an initial or *prior probability P(H)*. Let E' denote the observations made by the user. If E' completely confirms the proposition E (i.e. the *posterior probability P(E|E')* = 1), the posterior probability P(H|E') is equal to the *conditional probability P(H|E)*. If E' completely denies E (P(E|E') = 0), P(H|E') is equal to the conditional probability P(H|~E), where ~E denotes the negation of E. If E' neither suggests nor denies E (P(E|E') = P(E)), then P(H|E') = P(H).

Numerical values for P(H), P(E), P(H|E) and P(H|~E) are subjective estimates supplied by the expert, and if applied in a pure Bayesian formulation, they may produce inconsistencies. Duda et al.[1] discuss solutions to this problem. The solution adopted in Prospector is to define the general formula for P(H|E') as a piecewise linear interpolation using the three fixed points described above.

From the user's viewpoint, Prospector expresses degree of belief through the certainty measure introduced in the MYCIN system[14]. This *certainty value C(H|E')* measures the degree to which observed evidence E' changes the probability of the hypothesis H from its prior value P(H). C(H|E') is a number between -5 and +5, where -5 corresponds to H being definitely false, +5 corresponds to H being definitely true, and 0 corresponds to the "no-information" case where P(H|E') = P(H). Between these endpoints, there is a one-to-one linear relation between certainty and probability.

The main advantage of employing certainties is psychological. People often feel uncomfortable estimating probabilities, yet they are willing to say whether a piece of evidence increases or decreases the probability of a proposition with respect to its prior value. Thus, certainties are particularly useful for talking about relative probabilities.

So far, we have discussed measures of belief for simple propositions. It is often useful to consider propositions which deal with numeric quantities (for example, "the age of the rock"). To handle such a proposition, Prospector allows one to construct a *numeric input node* that describes a partition of the range of the numeric quantity into a set of intervals, and to specify a *variable strength rule* which has a conditional probability value for each of the intervals in the antecedent node of the rule. Procedures for computing P(H|E') for the consequent of a variable strength rule are outlined in Duda et al[3].

We have thus far discussed the effect of a single piece of evidence on an hypothesis. In the case of multiple rules affecting a single hypothesis, the assumption of conditional independence of the evidence is made, resulting in a simple method for computing the posterior probability of the hypothesis. This method, which is based upon Bayes' Rule of probability theory, is described in Duda et al[1].

Prospector also allows evidence to be combined using the logical combinations of negation, conjunction, and disjunction. The formula for negation is the classical one:

$$P(\sim E|E') = 1 - P(E|E').$$

Unfortunately, for conjunctive combinations

$$E = E_1 \text{ and } E_2 \text{ and } \ldots \text{ and } E_n$$

and disjunctive combinations

$$E = E_1 \text{ or } E_2 \text{ or } \ldots \text{ or } E_n$$

one needs to know the entire joint probability $P(E_1,E_2,\ldots,E_n|E')$, not just the conditional probabilities $P(E_i|E')$. The assumption of statistical independence leads to simplified formulas, but, in our experience, the results are usually too pessimistic for conjunction and too optimistic for disjunction. Prospector's heuristic solution uses the following formulas from the theory of fuzzy sets[15]:

$$\text{conjunction: } P(E|E') = \min \{P(E_i|E')\}, \quad i = 1, 2, ..., n$$

$$\text{disjunction: } P(E|E') = \max \{P(E_i|E')\}, \quad i = 1, 2, ..., n.$$

By combining multiple rules and logical combinations, one can capture relations between evidence and hypotheses that are much more complex than either extreme.

C. Context

It sometimes happens that one proposition refers to another, and one cannot meaningfully talk about the former in isolation. For example, one usually cannot consider a property of some object before the probable existence of that object has been established. Even when isolated reference is meaningful, it is often the case that an expert cannot provide strong rules unless the existence of some special situation has been established. To handle such situations, Prospector allows any proposition E_1 to be designated as a *context* for another proposition E_2 (in inference network terminology, a *context arc* links E_1 to E_2). The context is considered to be satisfied if the final posterior certainty value of E_1 falls within a range specified by the expert. If the context is not satisfied, Prospector in effect ignores proposition E_2. Since the context arc forces E_1 to be investigated before E_2, the context mechanism goes beyond the representation of factual knowledge into the area of *control.*

D. Semantic Networks

The meaning of the proposition represented by an inference network node can be encoded using a simple text string; e.g.: "a rhyolite or a dacite plug is present." However, there are advantages to a more explicit encoding of the meaning. In Prospector, this can be done using a set of n-ary relations; e.g.:

COMP-OF E1 (OR RHYOLITE DACITE)
FORM-OF E1 PLUG.

The first relation encodes the fact that there is an entity E1 composed of rhyolite or dacite; the second that E1 is in the form of a plug. If the elements of these relations are considered nodes and the relations expressed among them arcs, we can view sets of relations as a *semantic network*[16] which expresses the meaning of inference network nodes.

Semantic networks are used in conjunction with a *taxonomy* (another type of network which expresses subset-superset relations among the domain concepts referred to in the semantic networks) to detect whether two inference network nodes are semantically equal or whether one is a subset of another. This procedure is performed by a part of Prospector called the *Matcher*[10, 17]. If the Matcher detects either of these cases, Prospector can make inferences which are not explicitly encoded in the inference network, and it can also check for certain inconsistencies which might be produced by inferences.

E. Modes of Use

Prospector supports four primary modes of use: *consultation*, *batch*, *off-line*, and *compiled.*

Prospector is most often used as a consultation system with which the user conducts an interactive dialog, providing information on a specific situation to be analyzed. During a consultation, the system uses a *mixed-initiative* strategy to gather information. In *antecedent mode,* the user can tell the system about those aspects of the situation which he considers important. He does this using very simple English-like statements. The user's statements are translated to semantic networks which are processed by the Matcher in the manner described above. In *consequent mode,* Prospector chooses the top-level hypothesis which seems most appropriate to consider, then chooses the question (external evidence node) which seems most likely to confirm or deny that hypothesis. This series of decisions, called the *control strategy*, has been described in detail by Duda et al.[2] and Reboh[10]. Another important component of the consultation system is the explanation system, which at any time can provide the user with an easy-to-understand description of the conclusions which have been reached, along with information supporting those conclusions.

The batch mode of Prospector is normally used for testing knowledge bases. In this mode, the user's answers to Prospector's questions are stored on a disk file. At some later time (perhaps after modifications to the knowledge base), this file can be used by Prospector as the source of answers.

Prospector also supports off-line data collection through the ability to generate a questionnaire which lists all questions in a knowledge base. This is primarily used to collect a set of test cases which are used to quantify the quality of a knowledge base.

The compiled mode was developed to provide geologists with an ability to process graphical data. Using a digitizing tablet, the geologist traces the geological features requested by Prospector. After some preprocessing, each feature is encoded as a 128 x 128 array. Prospector is then run using point (1,1) from each feature array, then point (1,2), etc., to produce an array of scores for the top-level hypothesis, which is then color-coded and displayed on a graphics device. Arrays for lower-level hypotheses can also be computed and viewed to provide a type of explanation system. The map-processing system uses inference networks that have been compiled to simple machine-language instructions, thus reducing computational demands by four orders of magnitude.

III. The Design of Hydro

The Hydro system is an extension of Prospector designed to assist users of a large hydrological simulation model in estimating values for input parameters[11, 12]. The fundamental task performed by the Hydro software is the estimation of numeric quantities under conditions of uncertainty.

A. Basic Knowledge Representation

After several attempts at representing an uncertain numeric value using very simple schemes, the method settled upon is the use of a discrete probability distribution represented as a set of interval-probability pairs. Each interval is described by a lower bound and an upper bound, and its associated probability measures the likelihood that the value actually falls in that interval.

B. Value Networks

The knowledge base structures which manipulate distributions are referred to as *value networks*. The types of nodes in value networks are *tables, formulas, alternatives*, and the numeric input node discussed in Section II.B.

Table and formula nodes take one or more distributions as input and produce a single distribution as output. A table node with n inputs contains an n-dimensional table, where each entry in the table is an interval in which the output value is guaranteed to lie, given the combination of input intervals corresponding to the indices of the entry. A formula node is similar to a table node, the difference being that entries are computed using an analytic function stored with the node.

An alternatives node takes two or more distributions as input and produces one distribution as output by "multiplying distributions;" that is, the probability of each interval in the output distribution (before normalization) is the product of the probability of that interval based on the first input distribution, then based on the second, etc. Thus, alternatives nodes can be thought of as computing the conjunction of distributions.

Numeric input nodes are used as external evidence nodes for value networks. Also, the output of table, formula, and alternatives nodes can be fed into a numeric input node (in this case it is not treated as an external evidence node), which can then be connected to inference network nodes using variable strength rules (section II.B).

IV. Knowledge Engineering Efforts

The process of constructing a knowledge base is often referred to as *knowledge engineering*. Several substantial knowledge engineering efforts were undertaken during the Prospector project.

In the geological domain, twelve *prospect-scale models* (a *model* is an inference network with one top-level hypothesis) were developed, each encoding a particular class of mineral deposits. These models contained a total of 1566 inference network nodes and 1065 rules. In addition, twenty-three *regional-scale models* totaling 1683 inference nodes and 1370 rules were developed. Not all of these models have been developed to the point where they are ready for formal evaluation. However, systematic tests of mature models showed that they matched experts' evaluations of prospect sites to within a few percent[3, 9]. Moreover, a prediction of molybdenum mineralization in an area near Mount Tolman in the northwestern United States that was made using one of the three map-based prospect-scale models has been substantially confirmed by subsequent exploration[13].

V. Knowledge Engineering Tools

The magnitude of these knowledge engineering efforts necessitated the construction of software tools. The most developed of these tools is KAS[10], an interactive system for constructing, inspecting, editing, and testing knowledge bases.

Several other useful packages were also developed. The Probs program allows one quickly to test the effects of one level of rules on an hypothesis[8]. Another package of routines produces evaluation reports which compare Prospector's results using sets of input data to experts' evaluations of that data[9]. A network display package allows the display of inference and value networks on a graphics device[18].

VI. Conclusions

The Prospector project has demonstrated that it is possible to build large knowledge bases that can demonstrate a high level of decision making competence. Of course, knowledge bases must undergo thorough testing and tuning in order to achieve this. Nevertheless, the fact that knowledge engineering efforts have in many cases quickly produced reasonable results indicates that the techniques and tools employed provide a natural and effective way to encode judgmental knowledge** .

As others have noted, the rule-based approach provides a convenient way to express knowledge in small, modular units. It is not sufficient, however, to view a rule base as an unstructured collection of rules to which one can casually add or subtract. Viewing a set of rules as an inference network, on the other hand, provides a more structured, global picture of the rules. To an extent, the inference network approach promotes "structured programming" of rule bases.

It is clear that a variety of mechanisms are needed to express the expert's knowledge accurately. A primary requirement imposed by the geological domain is an ability to deal with uncertainty in both input data and in knowledge bases. A fruitful approach has been to be guided by the large body of work that has been done in probability theory, but to depart from it as necessary whenever efficiency or expressibility problems were solved more satisfactorily by other methods. For example, fuzzy logic combinations and the context mechanism were added to Prospector to improve expressibility. Semantic networks were used not only to increase expressibility but also to provide an efficient means of processing

**It is worth noting that early in the project some people felt that the judgmental knowledge of exploration geologists was unencodable.

(implicit) rules that do not involve uncertainty. The design of value networks was guided by probability theory, but several approximations were made in the interest of computational efficiency.

Also, it is clear that the knowledge acquisition tools built to assist in the laborious process of constructing knowledge bases are very important; however, experience in knowledge acquisition is at least as important. Knowledge bases built in the later phases of the project were constructed more than ten times faster than earlier ones.

Once encoded, knowledge bases can be used in a variety of ways. Prospector supports interactive, batch, off-line, and compiled modes, but others (e.g. tutorial) are conceivable and will certainly be developed as the need arises.

Obviously, Prospector has many limitations. If one takes an extreme view of the goals of expert system research, one may conclude that success will not be achieved until all the problems of artificial intelligence have been solved. This is a needlessly pessimistic conclusion. The primary goal of expert system research is *to provide tools that exploit new ways to encode and use knowledge to solve problems, not to duplicate* human behavior in all its aspects. Therefore, the following discussion of Prospector's limitations will be restricted to a few of the issues that directly impact this more modest goal.

A basic problem with the Prospector approach is that it is restricted to unidirectional inference; that is, a change in the certainty of an hypothesis has no effect on the certainty of its supporting evidence, although there are cases when expectations about the evidence should be changed. Also, Prospector does not handle mutually exclusive and exhaustive hypotheses properly; in particular, it will not reach a conclusion by ruling out alternatives unless rules are explicitly included for this purpose. In addition, even with KAS and other tools the process of building knowledge bases can still be very difficult. For example, assigning prior and conditional probabilities is by no means easy, and the assumptions on which the system is built sometimes force artificial adjustment of these values. Another basic problem is that Prospector is convenient only for encoding "shallow" heuristic models; it does not seem appropriate for causal or functional models (such as a nuclear reactor simulator or a model of traffic flow). Also, since Prospector is a research product, the current implementation suffers from the expected problems of speed, reliability, and portability.

In spite of these shortcomings, experience with Prospector and other expert systems shows that expert system research can lead to the development of high-performance programs. Equally important is the field's impact on the systematization and codification of knowledge previously thought unsuited for formal organization. Improved approaches to formalizing and managing knowledge are certain to be of importance to a wide variety of scientific and economic endeavors.

Acknowledgments

I wish to thank Richard O. Duda, Peter E. Hart, and Rene' Reboh not only for providing valuable comments on this paper, but also for developing and implementing many of the ideas described herein.

The research described has been supported by the National Science Foundation and the United States Geological Survey.

References

1. Duda, R. O., P. E. Hart, and N. J. Nilsson, "Subjective Bayesian Methods for Rule-Based Inference Systems," *AFIPS Conference Proceedings, Vol. 45,* AFIPS Press, AFIPS, 210 Summit Avenue, Montvale, NJ 07645, 1976, pp. 1075-1082.

2. Duda R. O., P. E. Hart, N. J. Nilsson, R. Reboh, J. Slocum, and G. L. Sutherland, "Development of a Computer-Based Consultant for Mineral Exploration," Annual Report SRI Projects 5821 and 6415, Artificial Intelligence Center, SRI International, Menlo Park, California, October 1977.

3. Duda, R. O., P. E. Hart, P. Barrett, J. G. Gaschnig, K. G. Konolige, R. Reboh, and J. Slocum, "Development of the Prospector Consultation System for Mineral Exploration," Final Report SRI Projects 5821 and 6415, Artificial Intelligence Center, SRI International, Menlo Park, California, October 1978.

4. Duda, R. O., P. E. Hart, N. J. Nilsson, and G. L. Sutherland, "Semantic Network Representations in Rule-Based Inference Systems," *Pattern-Directed Inference Systems,* Waterman, D. A. and Hayes-Roth, F., eds., Academic Press, New York, New York, 1978, pp. 203-221.

5. Duda, R. O., P. E. Hart, K. G. Konolige, and R. Reboh, "A Computer-Based Consultant for Mineral Exploration," Final Report SRI Project 6415, Artificial Intelligence Center, SRI International, Menlo Park, California, September 1979.

6. Duda, R. O., J. G. Gaschnig, and P. E. Hart, "Model Design in the Prospector Consultant System for Mineral Exploration," *Expert Systems in the Micro Electronic Age,* Michie, D., ed., Edinburgh University Press, Edinburgh, Scotland, 1979, pp. 153-167.

7. Duda, R. O., "The Prospector System for Mineral Exploration," Final Report SRI Project 8172, Artificial Intelligence Center, SRI International, Menlo Park, California, April 1980.

8. Duda, R. O., J. G. Gashnig, R. Reboh, and M. B. Wilber, "Operation Manual for the Prospector Consultant System," Final Report SRI Project 8172, Artificial Intelligence Center, SRI International, Menlo Park, California, January 1981.

9. Gaschnig, J. G., "Development of Uranium Exploration Models for the Prospector Consultant System," Final Report SRI Project 7856, Artificial Intelligence Center, SRI International, Menlo Park, California, March 1980.

10. Reboh, R., "Knowledge Engineering Techniques and Tools in the Prospector Environment," Technical Note 243, Artificial Intelligence Center, SRI International, Menlo Park, California, June 1981.

11. Gaschnig J. G., R. Reboh, and J. Reiter, "Development of a Knowledge-Based Expert System for Water Resource Problems," Final Report SRI Project 1619, Artificial Intelligence Center, SRI International, Menlo Park, California, August 1981.

12. Reboh, R., J. Reiter, and J. G. Gaschnig, "Development of a Knowledge-Based Interface to a Hydrological Simulation Program," Final Report SRI Project 3477, Artificial Intelligence Center, SRI International, Menlo Park, California, May 1982.

13. Campbell, A. N., V. F. Hollister, R. O. Duda, and P. E. Hart, "Recognition of a Hidden Mineral Deposit by an Artificial Intelligence Program," *Science,* Vol. 217, September 1982, pp. 927-929.

14. Shortliffe, E. H. and B. G. Buchanan, "A Model of Inexact Reasoning in Medicine," *Mathematical Biosciences,* Vol. 23, 1975, pp. 351-379.

15. Zadeh, L. A., "Fuzzy Sets," *Information and Control,* Vol. 8, June 1965, pp. 338-353.

16. Hendrix, G. G., "Encoding Knowledge in Partitioned Networks," in *Associative Networks--The*

Representation and Use of Knowledge in Computers, Findler, N. V., ed., Academic Press, New York, New York, 1979, pp. 51-59.

17. Reboh, R., "Using a Matcher to make an Expert Consultation System Behave Intelligently," *Proceedings, First Annual National Conference on Artificial Intelligence,* Stanford, California, August 1980, pp. 231-234.

18. Gaschnig J. G., J. Reiter, and R. Reboh, "Development and Application of a Knowledge-Based Expert System for Uranium Resource Evaluation," Final Report SRI Project 2225, Artificial Intelligence Center, SRI International, Menlo Park, California, October 1981.

AN ORTHODONTIC CASE STUDY INSTRUCTION SYSTEM BASED ON A RELATIONAL DATABASE SYSTEM

Y. Kanamori, J. Sugawara
School of Dentistry, Tohoku University, Sendai 980, Japan
and
Y. Masunaga
University of Library and Information Science, Ibaraki 305, Japan

Abstract: This paper describes the design and implementation of an expert system for use by students in orthodontics in studying cases. The system is based on a relational database system from which knowledge about past diagnoses and treatments is retrieved. Because of the very long period of treatment (e.g. two to ten years), the diagnosis logic has not yet been established in orthodontics. Therefore, an instruction system based on a knowledge/data base is essential.

1. Introduction

We define an expert system as a system which can simulate human experts/1,5,6/.

In orthodontics, it is very important to train students to diagnose cases based on the morophological characteristics of craniofacial skeltons. Such training is essential, because usually it takes two to ten years to treat cases, and therefore it is impossible for students to see the validity of their treatments in a two-year training course.

One of the characteristics of our orthodontic case study instruction system is that it is based on a knowledge base in which all valid and invalid past diagnoses and treatments are stored as orthodontic knowledge. The reason for this is that the logic of orthodontic diagnosis has not yet been established, and therefore a valid treatment in the past becomes the basis of treatment for a new patient.

Therefore, our system differs from other systems like MYCIN/1/. MYCIN is based on a firm diagnosis logic, so that its knowledge base consists of a set of rules and inference rules.

Other characteristics of our system are as follows: Since the system maintains an image (skull line drawings) database, it displays images to instruct students rather than just giving descriptive information. The system does not diagnose the input case. Rather, it infers the validity of the diagnosis and/or treatment plan proposed by a student by comparing it with analogous cases stored in the knowledge database.

2. System Overview

Figure 1 shows the overall architecture of our system.

Students may interact with the system in terms of three types of data: (a) result of a case analysis, (b) diagnosis, and (c) treatment plan. That is, a student may want to know (c) by showing (a) and (b) (we call this type-1 interaction), or know the validity of his treatment plan, i.e. (c), by showing (a), (b) and (c) (we call this type-2 interaction). Of course, other types of interactions are possible. However, these are not so significant from orthodontic point of view.

The expert system interprets the input. First, the system analyzes type-a data so that it can identify into which of nine possible categories (this classification has been established already in orthodontic research/2/) the given case falls. This becomes the basis of information necessary to retrieve analogous cases.

Both the knowledge base (KB) and the database (DB) are organized as tables. A table named DIAGNOSIS(CASE_NO., SEX, AGE_AT_DIAGNOSIS, BONE_AGE, DENTAL_AGE, BODY_HEIGHT, CAST_DATA_NO., ORTHOPANTOMO, HORIZONTAL_FACIAL_TYPE, VERTICAL_FACIAL_TYPE, FACIAL_SYMMETRY, X_RAY_FILM_NO.) is stored in KB, while a table named FEATURE(X_RAY_FILM_NO., GONIAL_ANGLE, CHIN_ANGLE, ..., BODY_LENGTH, ..., F1, ..., F80) is stored in DB. Eleven other tables are stored in DB.

To retrieve analogous cases from DB, QUEL-like relational database language has been implemented. Therefore, the core system translates a student's input into a relational query expression to retrieve analogous cases. If a type-1 interaction is initiated by a student, the system shows every treatment ever done by specifying the type-a and type-b input. If a type-2 interaction is initiated, the same action as for a type-1 interaction is first taken, and then the retrieved data is compared with the type-c input. If at least one case is retrieved which has an analogous treatment plan tagged as good, the system replies success to the student. It is difficult to define which two treatments are analogous. Our proposal is to use a two-dimensional treatment plan. It is a diagram indicating a plan of when and what treatments will be done. An example of this diagram is shown in Figure 2.

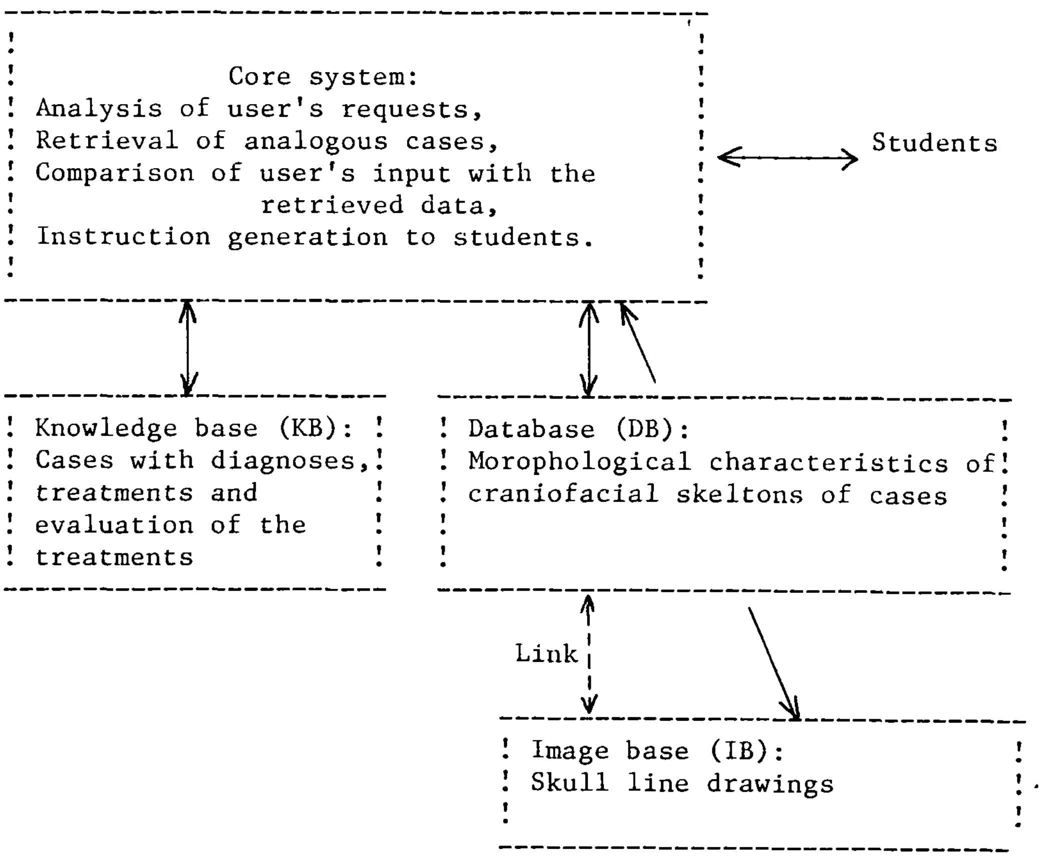
Core system:
Analysis of user's requests,
Retrieval of analogous cases,
Comparison of user's input with the retrieved data,
Instruction generation to students.
Students
Knowledge base (KB):
Cases with diagnoses, treatments and evaluation of the treatments
Database (DB):
Morophological characteristics of craniofacial skeltons of cases
Link
Image base (IB):
Skull line drawings

Figure 1. An overview of the system.

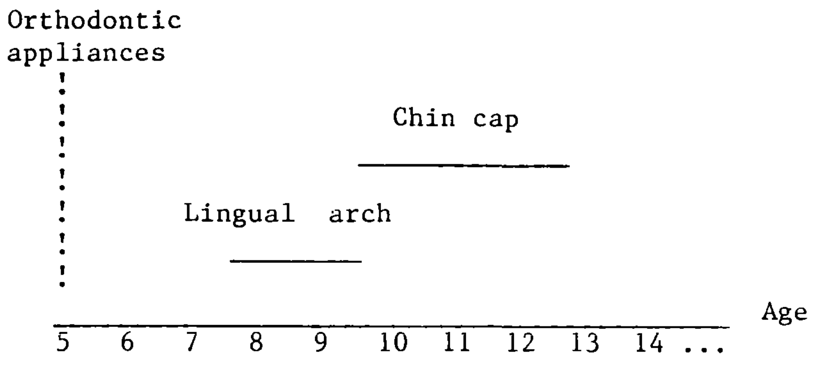

Figure 2. A sample of two dimensional treatment plan.

Since it is very effective to show skull line drawings of analogous cases directly to students, we have yet another type of database (Image Base, IB). In IB skull line drawings are represented as a collection of x-y coordinates of points. Of course, the (relational) DB and the IB are linked together. Among application programs, a program which can overlap certain skull line drawings to show the growth of a skull with certain reference points fixed is very useful. We have already implemented this function.

3. An Example Interaction with Students

Example: A student has a case of a boy who received his first examination at age nine. By inspection, the case seems to be anterior crossbite and deep bite. The student initially arrives at the following diagnosis and treatment plan.

Diagnosis: The case falls into the skeletal class III, and is short face and deep bite.

Treatment plan: During the first treatment period, a lingual arch will be used as an orthodontic appliance for correction. Next, a chin cap will be used. The student draws a two-dimensional diagnosis plan (or its equivalent) for the system (see Figure 2).

Then the student asks the expert system whether his treatment plan is valid (note that this is a type-2 interaction). The system behaves as shown below:

(1) It retrieves analogous cases (as orthodontic experts do in their brains). A case is analogous to the given one if the sex of the case is male, the first examination age is between eight and ten, the case falls into skeletal class III, the case is short face and deep bite, a lingual arch and a chin cap were used, and in order to ensure the data significance, the treatment was continued for more than five years, and improvement was recognized.

(2) It selects cases with stable prognosis from the cases retrieved on the previous step. The instruction given to the student is as follows: For each case retrieved, the system shows the overlapped skull line drawings to show the growth of the skull (by using the application program mentioned above), together with the two-dimensional diagnosis plan applied to the case and the validity of the plan.

(3) It selects cases with unstable prognosis. The system takes the same action as above in order to instruct the student on the errors of the diagnosis and corresponding treatment plan.

(4) It processes requests from the student. For example, the student may want to see certain cases again, or if the student is not satisfied with the instructions given by the system, he will ask the system to relax the

conditions the system adopted. For example, the student may want to relax the age of the cases to be between seven and eleven, or want to see cases which used other orthodontic appliances than a lingual arch and a chin cap. The system then will show other cases so that the student can compare his treatment plan with other types of treatment, and may finally understand the validity of his treatment.

4. Status and Future Plans

The following functions have already been implemented:

(1) A relational database management system, which runs on a TI 990/20 (Texas Instruments Inc.) mini-computer system, has been implemented on OS DX10 using PASCAL. This system supports a QUEL-like data manipulation language.

(2) A variety of application programs has been developed. The application programs involve the relational DB and the image base.

(3) Orthodontic data has already been organized as a relational database/3/. It has thirteen relations in third normal form. Three relations contain information about skull line drawings, and others are for diagnoses and treatments. The relation named DIAGNOSIS is the knowledge base (KB) of our system.

(4) In order to build skull line drawings database (IB), programs for processing input skull line drawings are necessary. We have already implemented an image processing system for this purpose/4/. This system also runs on TI 990/20, and is implemented in FORTRAN 77. A facsimile input is provided for the system.

Our future plans include the following:

(1) Implementation of the core system (Fig.1) on top of our relational database management system. PASCAL will be used as the implementation language.

(2) Facial and oral color photographs are sometimes essential in case diagnosis and treatment planning. However, these data have not yet been integrated into the database. We plan to use optical disks to store these data, and integrate the disk data management system into the present relational database system. Integration will be done by homogenizing the heterogeneous data at the relation level.

/References/

/1/ Shortliffe,E.H., Computer-based medical consultations: MYCIN, Elsevier (New York), 1976.
/2/ Sugawara,J., Kanamori,Y., and Sakamoto,T., Analysis of mandubular form in orthodontics, Proc. of the third world conference on medical informatics, pp.1168-1172, 1980.
/3/ Kanamori,Y., Masunaga,Y., Kido,K., and Noguchi,S., Design of a database system for skull line drawings processing in orthodontics based on the relational model, ibid, pp.1154-1158.
/4/ Kanamori,Y., Kido,K., Sugawara,J., and Sakamoto,T., Pattern recognition system for line drawings image traced from roentgen cephalogram in orthodontics, Proc. MEDIS 78, B-5-8, 1978.
/5/ Kulikowski,C.A., Artificial intelligence methods and systems for medical consultation, IEEE PAMI, 2-5, pp.464-476, Sept. 1980.
/6/ Nau,D.S., Expert computer systems, IEEE computer, 16-2, pp.63-85, 1983.

EXPERT SYSTEMS FOR BUSINESS APPLICATIONS

- A RESEARCH PROJECT AT NEW YORK UNIVERSITY - *

Yannis Vassiliou, Matthias Jarke, James Clifford

Computer Applications and Information Systems
Graduate School of Business Administration
New York University

1.0 INTRODUCTION

The purpose of our current research in expert systems and databases is to evaluate the potential for the use of expert systems in a business environment. The first major objective of the project is to develop a working, prototype expert system in an actual functional area of business. A second major objective is to investigate in general the possible advantages of combining an expert system with a relational database management system.

In this paper, the major components of the project are described. Emphasis is placed on the database management system--expert system interaction, where implementation is underway. First, the selected application is described. Second, the implementation strategies and tools are presented. Finally, more general research topics and the current project status are addressed.

2.0 APPLICATION

The first component of the project is the business application which is to be supported by the expert system. In selecting an application, the following criteria were used:

* This work is carried out as part of a joint research study being conducted with the IBM Corporation. The project researchers from New York University are: H. Lucas (project manager and coordinator), J. Clifford, M. Jarke, W. Reitman, T. Sivasankaran, E. Stohr, Y. Vassiliou, and N. White.

1. Expertise exists and is available.

2. The business problem is faced by an organization, rather than an individual.

3. There is a sufficient amount of data needed in decision making to justify the use of a database management system.

4. As in many typical business problems, the invocation of operations research and computation models is required.

5. The task of the application is non-trivial, yet well-bounded and easily expandable.

After considering several candidate applications, the decision was made to develop an Expert System (ES) for customizing insurance policies - an ES that meets the criteria set forth.

There are two major reasons why there exists a scope for such an ES. First, insurance companies sell only standardized products and therefore may not offer special benefits uniquely needed by a customer. It is not that companies cannot make adjustments on an individualized basis, but it is the amount of time, effort and the expertise that is required in computing the premium and assessing feasibility that discourage them to sell policies on a customized basis.

Second, forming the right benefit mix is more of an art than a science, exercised by able insurance agents. Capturing their informal rules into an expert system can be helpful to agents, and can result in better customer satisfaction and increased sales.

The expert will be assisting the insurance sales agent while recommending policies to a customer. For this it should be able to:

- generate unique mixes of benefits and compute premiums,
- assess feasibility,
- present the result and explain the reasoning process,
- allow for modifications of the reasoning process.

The knowledge base of this expert system includes formal rules (from actuarial science, finance, insurance law), and informal heuristic rules (e.g. generation of policies, typical customer requirements, identifying legal and corporate constraints). A large amount of factual knowledge (database) is necessary (customer data, tables for mortality, interest, etc.).

A small demonstration prototype has been built which shows the feasibility of the application but currently still lacks the interfaces to a commercial DBMS and to a mathematical subroutine library.

3.0 IMPLEMENTATION TOOLS

The expert system is rule-based [Nau 1983]. The programming language Prolog [Kowalski 1979] was selected for the implementation of the ES. A large relational database system that can be accessed through SQL [Chamberlin et al 1976] is used. Interfaces to the expert system for knowledge acquisition and consulting sessions will be implemented in ILIS - a program for developing language processors [Sowa 1981]. The hardware configuration includes an IBM 4341 running under VM/CMS.

4.0 DBMS--ES INTERACTION

The interaction between the two systems can be viewed in two complementary ways: Enhancing an ES with DBMS facilities, and enhancing a DBMS with ES techniques and features.

4.1 ES Enhancements With DBMS Facilities

In a rule-based expert system, the "inference engine" uses a set of rules (the knowledge base) and a collection of specific facts (database) to simulate the behaviour of a human expert in a specialized problem domain.

Expert system databases are typically conceptually complex (i.e deal with many entities and relationships), but differ from commercial databases in that they tend to be smaller in terms of actual data occurrences. On the other hand, expert systems for business applications may require time-varying subsets of very large databases.

Four strategies for establishing a cooperative communication between the deductive and data components of an expert system have been identified [Vassiliou et al 1983]. The spectrum of possible enhancements of an expert system with data management facilities is essentially a continuum.

Starting from elementary facilities for data retrieval we progress to a generalized DBMS within the expert system, to a 'loose' coupling of the ES with an existing commercial DBMS, finally, to a 'tight' coupling with an external DBMS.

Expert system designers may opt to choose one configuration over another depending on data volume, multiplicity of data use, data volatility characteristics (how frequently data is changed), or data protection and security requirements. Regardless, in a careful design these enhancements are incremental, allowing for a smooth transition from a less to a more sophisticated environment.

The first stage in the development, elementary data management facilities, is almost automatic when Prolog is used.

The implementation of a generalized DBMS in Prolog reduces the need for detailed data handling facilities in each part of the ES leading to the second stage of the development.

Having this generalized DBMS implemented, it becomes feasible to interface effectively with existing external databases, typically managed by separate commercial DBMSs.

Loose coupling (stage 3) implies the existence of a communication channel between the external DBMS and the ES, and the ability to extract a snapshot from the existing database and storing it as an ES database. This is a static process, occurring before the ES session. This stage reaches its limits where the set of database data is not known in advance, or is so large that only parts of the data can be stored in the internal database at any given time.

Tight coupling, then, refers to a more dynamic use of the communication channel between the two systems. The consequence of such dynamic use is that the external database becomes an "extension" of the ES database. This fourth stage of development presents the most challenging research issues. There is currently a working implementation [Vassiliou et al 1983] that is based on an amalgamation of Prolog (the ES-language) with its meta-language (the reflection principle of [Weyhrauch 1980]).

Research topics under investigation for 'tight' coupling include the translation of Prolog statements to SQL queries, an optimization of Prolog goals before this translation, the control primitives of the communication channel's use, and the exploration of other strategies for implementing the coupling of an ES with a relational DBMS.

4.2 DBMS Enhancements With ES Techniques And Features

When compared to requirements for advanced business applications, such as decision support for managerial users and office automation, current DBMSs display several weaknesses. For instance, DBMSs have been criticized for not supporting high-level languages, query languages that allow for "easy" and "intelligent" use of the database. This inability is often attributed to the absence of reasoning capabilities and deduction mechanisms from DBMSs. In addition, no support of operations performed in the context of other operations, of a certain application, or of a specific user, and no support of complex operations (e.g. mathematical computations) directly in the query language, are offered in current DBMSs. Finally, DBMSs can only enforce very simple integrity constraints, thus the accuracy of the database is often suspect.

At New York University, we are exploring how expert system technology can be used for some of these DBMS problems. We are employing a two-stage approach [Jarke and Vassiliou 1983].

Intelligent Database Use refers to making query languages functionally more powerful without making them more difficult to use. The ES supports intelligent database use by allowing one to state general horn clause rules (possibly with recursion) in addition to simple relational queries. Features that can be supported this way include:

- definition of generalized views with parameters
- query languages with deductive capabilities
- concepts of higher-level data models
- interaction of database and mathematical models

This stage can easily be added to an existing DBMS if 'tight' coupling mechanisms are available. One example of data model extensions under study in our group is the inclusion of time concepts [Clifford and Warren 1983].

Intelligent Database System Operation refers to safer and faster execution of read and write transactions. Our approach is to use the inference engine of the ES to implement transformation strategies such as proposed in [Jarke and Koch 1982, 1983]. Although these strategies work on a high-level query representation, their most efficient implementation will require changes to the DBMS itself. Two areas of applying rule-based methods can be distinguished:

- An ES for write transaction execution (consistency checking, referential integrity, etc.).
- An ES for query evaluation (evaluation of complex queries with quantifiers and aggregates, logic-based query simplification, semantic query evaluation, recognition of query context, etc.).

5.0 CONCLUDING REMARKS

Our discussion shows that both directions of interaction between databases and expert systems are closely related. For example, the translation of ES goals to DBMS queries would greatly benefit from a query optimization ES on top of the DBMS. Therefore, we expect a major advantage from researching these questions simultaneously. The ultimate goal is an advanced decision support system that integrates database management, model management, and expert system technology in a single architecture.

REFERENCES

1. Chamberlin, D.D., et al, 'SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control', IBM Journal of Research and Development, Vol. 20, 6, 1976, 560-574.

2. Clifford, J., Warren, D., "Formal Semantics for Time in Databases", ACM Transactions on Database Systems, Vol. 8, 2 (1983).

3. Jarke, M., Koch, J., "A Survey of Query Optimization in Centralized Database Systems", NYU Working Paper CRIS#44, GBA 82-73 (CR), November 1982.

4. Jarke, M., Koch, J., "Range Nesting - A Fast Method to Evaluate Quantified Queries", Proc. ACM-SIGMOD Conf., San Jose, May 1983.

5. Jarke, M., Vassiliou, Y., "Coupling Expert Systems with Database Management Systems", Artificial Intelligence Applications for Business (W.Reitman, ed.), Ablex, to appear 1984.

6. Kowalski, R.A., Logic for Problem Solving, North-Holland Elsevier, New York, 1979.

7. Nau, D., "Expert Computer Systems", Computer, February 1983, 63-85.

8. Sowa, J., "Interactive Language Implementation System", IBM Systems Research Institute, October, 1981.

9. Vassiliou, Y., Clifford, J., Jarke, M., "How does an Expert System Get Its Data?", NYU Working Paper CRIS#50, GBA 82-26 (CR), extended abstract in Proc. 9th VLDB Conf., Florence, October 1983.

10. Weyhrauch, R., "Prolegomena to a Theory of Mechanical Formal Reasoning", Artificial Intelligence, Vol. 13, 1980, 133-170.

Basic Decisions about Linking an Expert System with a DBMS: A Case Study

Gilles M. E. Lafue

Schlumberger-Doll Research
Old Quarry Road
Ridgefield, CT 06877

1. Introduction

It has been recognized that expert systems (ES's) and Data Base Management Systems (DBMS's) could benefit from one another: see, for example, [Reiter1983]. The existence of common interests in the two fields should be no surprise. Their nature is such that they have both developed from their tender years a strong interest in data/knowledge modelling and in inference making supported by that modelling. *Data models* have been developed in databases, and *knowledge representation systems* in AI. To the extent that data models include relationships between data and procedures within the management of data, they show important similarities with knowledge representation systems. The two approaches, however, have not necessarily placed their emphasis on the same aspects. In AI, for instance, inference making has been on the forefront and data with variable structure has been considered more important than data with efficient storage and access. In constrast, DBMS's have assumed more voluminous and more permanently structured data.

It is precisely based on these different viewpoints that exchanges between the two fields can be expected to be fruitful. For instance, AI techniques could be used to augment the reasoning which can be made about databases (DB's), or to provide them with more natural language interfaces. It seems clear too, that increasingly, ES's will need to manage large amounts of data. This is needed for many realistic applications, e.g., in CAD where the nature of ES's is to generate large amounts of data [Lafue1983], for supporting acceptable justifications of the ES's conclusions or for supporting attempts at machine learning.

This paper discusses basic decisions about the linking of a particular ES to a DBMS. It concerns both inference making and data management, although it emphasizes the latter. The starting point of the discussion is the claim often made in the recent past that there is an obvious way to unify ES's and DBMS's, namely using PROLOG, or, more generally, logic programming. This claim is generally justified by the fact that logic programming offers both an inference engine and a natural connection with the relational data model [Codd1970]. The case discussed in this paper, however, shows an example of an ES for which this claim does not hold, simply because the inference that is needed is not that of logic programming.

Data management considerations center around the choice between connecting the ES to an existing DBMS, i.e., building a bridge between the knowledge representation system it uses and an existing DBMS, vs. extending its knowledge representation system into a DBMS. These considerations are discussed both from a data modelling perspective and from a systems and efficiency perspective. From a data modelling perspective, the choice is beween the relational data model (as the most advanced among the data models of widely available commercial DBMS's) and the data model of the knowledge representation system used by the ES. This model is richer than the relational one and some of its salient features are presented. The alternative of translating this model into the relational one in order to hook up with an existing DBMS would be justified mostly as a way to cut down efforts and complexity and if it does not cause too much inefficiency. It turns out that particularities of our case would in fact increase inefficiency and not necessarily reduce complexity.

The next section introduces the context in which these choices arise. Then, inference making considerations are presented. The discussion of DBMS alternatives follows, both from a data modelling and from an efficiency viewpoint.

2. The Context

Schlumberger is engaged in the acquisition and interpretation of geophysical measurements made in boreholes, or wells, aimed at producing oil. These measurements are made by tools which are lowered into the boreholes, and are recorded as *logs*. The interpretation of these logs is aimed at determining the formations traversed by the boreholes, e.g., their geological and geophysical characteristics.

Such interpretation is (partly) based on a number of programs. One of these programs is an ES, called the *Dipmeter Advisor*, which derives from conductivity measurements made down a borehole a description of the geology around the borehole [Davis1981], [Smith1983b]. The Dipmeter Advisor is rule based. It is written in Interlisp and runs on Xerox 1100 Workstations. Other programs have been written in Fortran and currently reside on VAXes under VMS.

The notions of geology and geophysics manipulated by these programs are often embedded in rules or in executable code. Furthermore, these programs are generally découpled from one another, when in fact, they often manipulate related notions and could use each others findings in a cooperative manner. It was therefore decided to provide them both with more and better structured information (deeper knowledge) and with broader information about other measurements and interpretations (broader knowledge). This led to a system which monitors the access to various log files and the control of various programs. These accesses and activations being driven by models of log interpretation, this system can be seen as a log interpretation advisor. The files are stored on VAXes under VMS and the programs execute either under VMS or on the Xerox 1100 workstations. The monitoring is done on the Xerox 1100 workstations.

The basic tool to support this inflation of information and programs to control is an object-oriented language called STROBE [Smith1983a], [Schoen1983]. It resembles other such systems developed either in AI and then called knowledge representation systems (e.g., UNITS, Loops [Bobrow1981]), in databases and sometimes referred to as semantic data models (e.g., SDM [McLeod1982], TAXIS [Mylopoulos1980]), or in programming languages (e.g., SmallTalk [Goldberg1983]). Basically, it encapsulates into *objects* data and procedures pertaining to the same semantic entities. It also encourages the organization of objects into hierarchies in which an object can be the *generalization* and/or the *specialization* of other objectss. For instance, an object for employee is a specialization of an object for person and a generalization of an object for secretary. An object inherits the attributes of its generalizations. For example, a secretary has a salary because s/he is an employee and employees have a salary. In STROBE, an object can have several generalizations. Attributes can also be used to implement arbitrary relationships between objects as in Abrial's model [Abrial1974] or Entity-Relationship based data models [Chen1976]. STROBE is written in Interlisp for the Xerox 1100 workstations. More will be said about STROBE's data model later.

STROBE objects are used to represent programs and their executions, files of log measurements and a number of notions pertaining to log interpretation, e.g., classification of geological features, descriptions of wells, of tool characteristics. Some objects are generated as the system is modified, not as it is used. This is the case, for example, of objects representing interpretation programs or geological classifications. Such objects correspond to a database schema. Other objects are generated as the system is used and correspond to a database extension. Examples of such objects are those for program executions, for files of measurements, for instances of geological features or for the manifestations of these features in individual wells.

It has been decided to keep dynamically generated objects as a history of automatic log interpretation because it is necessary in order to explain the results of intepretations of individual wells, and, more generally, to improve interpretation methods. Another important reason is to support the modelling of oil fields which can only be done with information from several wells. The number of objects to keep is such that it requires database facilities.

3. Inference

One emerging approach to linking ES's and DB's stems from the resurgence of first order logic and the advent of logic programming [Gallaire1978], [Gallaire1981], [Kowalski1979]. This is exemplified by the rising popularity of PROLOG [Colmerauer1983]. See, for instance [Vassiliou1983], [Parsaye1983], [Zaumen1983]. The conceptual basis for this is that the logic statements of PROLOG, i.e., Horn clauses, can be considered either declaratively or procedurally.

Considered declaratively, they express queries or integrity constraints about a database, or even the database itself. That database can be seen as a relational database. It consists of statements sometimes referred to as *facts*, which are degenerate clauses of the form ->P. Queries are degenerate clauses of the form P-> and can either be open formulas, i.e., containing free variables, or closed formulas. Treated procedurally, queries are *goals* to be proved or refuted with theorem-proving techniques, e.g., resolution's unification and top-down depth-first search with backtracking as in PROLOG. The proof of an open query returns the instantiations of its free variables in the current database. The proof of a closed query returns a boolean value. Integrity constraints, or *assertions*, or *rules*, are clauses, i.e., of the form P,Q,R->S. Procedurally, they are used to guide theorem-proving as follows: since a rule can be read as "S if P, Q and R", then in order to prove S, it suffices to prove P, Q and R.

The apparent appeal of logic programming as the tool to unify ES's and DB's is that it comes with an inference mechanism and naturally connects to relational databases. On the other hand, this strength can also be a weakness to the extent that it is imposed. This may be the case when the type of inference provided by a logic programming system and that needed by an application do not agree, or when the relational data model is not an obvious choice.

Logic programming, as implemented in PROLOG, is top-down (or backward) inference making, i.e., it aims at proving goals. It does so by trying to refute goals, traversing databases of facts and rules and looking for inconsistencies between facts and instantiations in the rules of the goal refutations and of their implications. In addition, PROLOG imposes its top-down search strategy, i.e., depth-first with backtracking, although it also offers ways to restrict backtracking.

In constrast, log interpretation is based on bottom-up (or forward) inference. The Dipmeter Advisor's inference engine, for instance, is forward chaining. It searches facts and rules and when a match is found or deduced between facts and the antecedents of a rule, an instance of the rule's consequent is generated and included in the database. It infers the presence of geological features from the recognition of patterns of signals. Like signal processing in general, log interpretation is data-driven. Its aim is the generation of plausible conclusions from a mass of possible ones. The number of possibilities would make attempts to prove or refute the presence of every single one prohibitively expensive.*

* In reality, human log interpreters tend to mix a bit of top-down inference in their interpretation. For instance, they sometimes form some hypothesis based on weak evidence, and then look for signal patterns confirming this hypothesis.

It should be underlined that this mismatch between logic programming and log interpretation does not at all imply a general repudiation of logic, logic programming, top-down inference, or PROLOG, but only a recognition that different types of inference may have different places and roles in the marriage of an ES and a DBMS. For example, logic programming's top-down inference can be used in query evaluation.

4. The DBMS Alternatives

There are basically two ways to conceive the connection of a DBMS to a given ES: take an existing DBMS and build the bridge between the two, or develop a new DBMS to suit the ES. If one is interested in general-purpose DBMS's, then relational DBMS's are likely candidates since they respond to some of the obvious deficiencies of earlier DBMS's and are becoming widely available commercially. The success of the relational data model is largely due to its simplicity and generality. However, this is not necessarily sufficient if there is an alternative for a richer data model and if the connection to an existing relational DBMS is awkward.

This section discusses the alternatives from two view points: (i) data modelling and (ii) systems programming and efficiency. From the former view point, we compare the knowledge representation system in which the ES is implemented, STROBE, with the relational model. From the latter view point, we discuss some implications of linking the ES with a particular relational DBMS, INGRES [Stonebraker1976], [Ingres1982], and we contrast them with the alternative of basing a DBMS on STROBE's object oriented features.

4.1. Data Modelling Considerations

In general, it is fair to say that STROBE offers a richer data model than the relational model. Below, the salient data modelling features of STROBE which are not explicitly part of the relational model are presented, and their possible mapping into the relational model is discussed.

4.1.1. Four Levels of Data Representation

Whereas the relational data model offers three levels of data representation (i.e., databases, relations and attributes), STROBE offers four levels of data representation: *knowledge bases* which are collections of related *objects*, which in turn have *slots* (or attributes), which in turn are further described by *facets*. Typical facets of a slot are the slot's value and its data type (e.g., integer, pointer to another object, bit map, ..). The user can add any facet, for example, to specify the types of objects the slot can point to, or the minimum and maximum number of such objects, or to specify the units in which the value of a slot representing a dimensioned quantity is expressed.

There are two kinds of STROBE objects: *classes* which can be instantiated, and *individuals* which cannot. A class can have a *progeny* made either of other classes or of individuals. A collection of STROBE classes corresponds to a database schema, and a collection of individuals to a database extension.

Transforming a STROBE schema into a relational one would mean compressing the four levels of STROBE into the three levels of the relational model. Conceptually, the STROBE facets are the missing level in the relational model and the analogy is naturally drawn between objects and relations on one hand, and between slots and attributes on the other. However, if one wants to keep the facet information in a relational schema, the compression of STROBE classes into relational schemas would result in a decomposition of objects into relations. This is especially true if the relational normalizations are respected. For example, a M x N relationship between two objects implemented with one slot in each object would become a relation with the facets of the two slots as attributes.

4.1.2. Generalization and Inheritance.

There are two kinds of things an object with generalizations (or classes) can inherit from its generalizations: (i) a set of attribute definitions, and (ii) default values for these attributes. STROBE handles both with the notion of class. The slots of a class need not have values. If they do, their values are inherited by their progeny as default values.

Inheritance of attribute definitions exists in the original relational model only from a relation to its tuples, not really from one relation to another. For instance, in order to implement the fact that employee is a specialization of person, one could explicitly add an integrity constraint stating that any employee tuple must correspond (e.g., have the same social security number) to exactly one person tuple. Not only is this done outside the data structure, but it is not sufficient to go down the generalization hierarchy, e.g., to know that a person can be an employee, or an employer, or unemployed, ... (unless a class and its whole progeny are stored in the same relation). The absence of this property has long been recognized as a major hindrance, and a proposal has been made to remedy it [Smith1977]. However, this proposal requires the addition of an operator to the relational algebra and has not been implemented in available relational DBMS's.

The inheritance of default values basically requires the data model to allow constructs with a variable number of attributes. Attributes which have a default value in a generalization only need to appear in the specializations of this generalization where the default value is overridden. This contradicts the (conceptual) model of relations as tables.

4.1.3. Uniform treatment of actual and virtual data.

In STROBE, facets can either contain data or executable code, such as a procedure name or a Lambda expression (since STROBE is written in Interlisp). This facility allows smooth integration of data and code in objects, thus hiding from the user whether data is actual or virtual. Furthermore, the access mechanism is the same in both cases. The contents of facets are accessed by sending them *messages*. A message is sent to the facet of a slot of an object. If a facet contains some data, that data is returned. If it contains a procedure name or definition, that procedure is executed and its value returned. This can be used to retrieve or recompute data, to perform checking, or to take arbitrary actions.

It is also possible in STROBE to have the code stored in one facet of a slot automatically activated before or after another facet of that slot is accessed or updated. This corresponds to triggers or alerters [Buneman1979]. In STROBE, the automatic activation of procedures attached to data can be specified to take place before or after that data is accessed or updated, and several procedures can be pipelined as a result of such an event.

4.1.4. User-defined data types.

STROBE slots can be of any user-defined data type. A data type is in fact implemented as an object whose main purpose is to contain the definition of the operations and integrity constraints for the data type. This is not generally provided by relational DBMS's, but is left to the programming language within which query languages can be embedded. In such environments, generic operations, i.e., operations of the data model's Data Manipulation Language, are defined and performed under the DBMS's control, whereas application-dependent data operations are performed under the programming language's control. This issue of dual environment and control is revisited in the next section.

4.2. Systems and Efficiency Considerations

STROBE currently lacks most of the secondary memory management facilities traditionally offered by DBMS's, such as file management, concurrency control, protection and crash recovery. Moreover, the only DBMS for the Xerox 1100 workstations we know of at this point is still experimental and runs under Mesa [Cattell1983]. An alternative to extending STROBE into a DBMS would consist of using a DBMS running on VAX under VMS, since as mentioned earlier, our ES accesses files and activates programs under VMS on VAXes from Interlisp-D on Xerox workstations. We shall briefly investigate this possibility by using INGRES as an example, and contrast it with making STROBE into a DBMS.

The scenario for accessing data in INGRES from STROBE would resemble the following. Suppose a query language has been designed, possibly embedded in Interlisp, which produces queries in INGRES' query language, QUEL. Since STROBE manipulates objects and INGRES manipulates relations, this would first require mapping STROBE schemas into relations, and the translation into QUEL would use this mapping. Queries are sent to VMS and submitted to INGRES as standalone queries. They are interpreted and executed by INGRES . The results are sent back to the workstation and transformed back into STROBE objects. User-defined operations, either datatype operations or procedures attached to data, are then applied to the results. This still leaves many questions raised by STROBE data model unanswered: How is STROBE generalization of classes represented in INGRES? How are default values handled? How is the activation of procedures contained in facets handled?

A major drawback of this alternative is the particularity observed earlier about embedding query languages into programming languages that operations on data are performed differently depending on whether they are application dependent or generic (i.e., provided by the DBMS's data model and therefore, application-independent). Generic operations would execute under INGRES control, and application-dependent operations under Interlisp/STROBE.

One consequence of this dual execution environment is that the execution of generic database operations would be interpreted whereas application dependent operations could be compiled. The interpretation of data accesses is often justified by considerations of data independence: it allows data to change without affecting the code accessing it. Interpretation is therefore generally recommended in case of frequently changing data. In case of stable data, however, significant increase in speed of execution can be gained by compiling data accesses. A compromise, similar to System R's approach, consists of compiling data accesses, but whenever their data changes they are flagged as outdated, and recompiled at the first attempt to execute them. This would be impossible in our case due to the forced interpretation of QUEL queries

The inconvenience of dual execution control would be all the more regrettable that the object-oriented nature of STROBE is an invitation to treat all data operations, generic or not, in a uniform manner. Generic database operations would be attributes of a high level object representing the class of objects stored on secondary memory. Every object susceptible to be stored on secondary memory would be a specialization of that high level object and, consequently, would inherit the generic database operations. It would then have the option of refining these operations and/or of adding its own. Also, the object(s) containing the generic data operations would encapsulate both the external representation and the internal representation of the data model. The external representation would consist of the availability of these operations, and the internal representation would consist of their implementation. It would make it easier to change the data model implementation without affecting its users.

In summary, the connection between STROBE and INGRES does not appear natural. This is hardly a surprise. If Stonebraker in a retrospective on the design of INGRES thinks that attempting to glue together C and QUEL is "rather like interfacing an apple to a pancake" [Stonebraker1980], then what kind of an interface would the one between STROBE and QUEL be?

Note, however, that this does not necessarily preclude some relational implementation for STROBE databases. Such an implementation could be useful, especially in order to keep open communications with the outside world. For instance, it is probable that many database machines will be based on the relational model, since it is the most common of the set-oriented data models. Relational database machines will implement relational algebra operators, that is, lower level operators than for instance, QUEL. One reason INGRES does not fit our plan is that it would not be a backend machine, but a backend DBMS, with all the implied overhead and duplication of efforts and the imposition of a fixed query evaluation strategy. The coupling of an ES on a Xerox workstation and a backend database machine could resemble the architecture described in [Kellogg1983].

5. Conclusion

This paper illustrates a case of selecting a DBMS for an ES. Two (related) aspects were discussed: inference making and data management. First, this case was presented as an example where the unification between databases and ES's offered by logic programming would not necessarily work because logic programming's top-down inference is not an obvious approach.

Then, the DBMS alternatives were presented as hooking up with an existing DBMS vs. extending the ES's knowledge representation system (STROBE) into one. For the former alternative, relational DBMS's in general, and INGRES in particular, were considered. The question was addressed both from a data modelling viewpoint and from a systems and efficiency viewpoint.

It was felt that STROBE has some desirable data modelling features which should be kept. The mapping between STROBE and the relational model is not straightforward (although feasible). However, the connection between STROBE and a relational DBMS such as INGRES would be acceptable only if it does not get too complicated by systems issues and slowed down by inconsistencies between the two systems. This is apparently not so. A major source of trouble would be the fact that data operations would execute in different environments depending on whether they are generic or application-dependent. Alternatively, the object-oriented nature of STROBE is viewed as an opportunity to treat all data operations uniformly.

In conclusion, it was decided to extend STROBE into a DBMS. This solution is in line with the continuing efforts towards richer data models. It also agrees with proposals to implement DBMS's in object-oriented languages, e.g. [Baroody1981]. Finally, it is consistent with other attempts to extend existing programming languages into DBMS's. See, for instance, the approaches consisting of making Algol's or Ada's heaps permanent [Atkinson1982], [Hall1983]. In particular, making Ada's heap permanent is to be contrasted with the attempt to combine Ada with an existing DBMS (Daplex) [Smith1981]. In general, we see this approach as an opportuniy to study the potential contributions of data organization to enhance inference.

Acknowledgements

Comments, suggestions and fruitful discussions with Reid Smith, Peter Will and Stan Vestal have significantly contributed to this paper and are appreciated. The editor made useful suggestions about an earlier version of the paper.

References

Abrial1974. J.R. Abrial, "Data Semantics," in *Data Base Management*, ed. Klimbie & Koffeman,North Holland (1974).

Atkinson1982. M. Atkinson, K. Chisholm, and P. Cockshott, "PS-Algol: An Algol with a Persistent Heap," *SIGPLAN Notice* **17**(7)(July 1982).

Baroody1981. A.J. Baroody and D.J. deWitt, "An Object-Oriented Approach to Database System Implementation," *ACM Transactions on Database Systems* **6**(4)(Dec. 1981).

Bobrow1981. D. Bobrow and M. Stefik, "The LOOPS Manual," Xerox Palo Alto Research Center (1981).

Buneman1979. O.P. Buneman and E.K. Clemons, "Efficiently Monitoring Relational Databases," *ACM Transactions On Database Systems* **4**(3)(Sept. 1979).

Cattell1983. R.G.G. Cattell, "Design and Implementation of a Relationship-Entity-Datum Data Model," Xerox Palo Alto Research Center (May 1983).

Chen1976. P.P. Chen, "The Entity-Relationship Model. Toward a Unified View of Data," *ACM Transactions on Database Systems* **1**(1)(March 1976).

Codd1970. E.F. Codd, "A Relational Model of Data for Large Data Banks," *Communications of the ACM* **13**(6)(June 1970).

Colmerauer1983. A. Colmerauer, "PROLOG in 10 Figures," in *Proceedings of International Joint Conference on Artificial Intelligence*, , Karlsruhe W. Germany (1983).

Davis1981. R. Davis, H. Austin, I. Carlbom, B. Frawley, P. Pruchnik, R. Sneiderman, and J.A. Gilreath, "The Dipmeter Advisor: Interpretation of Geologic Signals," in *Proceedings of International Joint Conference on Artificial Intelligence*, , Vancouver Canada (1981).

Gallaire1978. H. Gallaire and J. Minker, *Logic and Data Bases*, Plenum Press, New York (1978).

Gallaire1981. H. Gallaire, J. Minker, and J.M. Nicolas, *Advances in Data Base Theory*, Plenum Press, New York (1981).

Goldberg1983. A. Goldberg and D. Robson, *Smalltalk-80 The Language and its Implementation*, Addison-Wesley (1983).

Hall1983. A.V. Hall, "Adding Database management to Ada," *SIGMOD Record* **13**(3)(April 1983).

Ingres1982. Ingres, "Ingres Manual," Relational Technology, Berkeley, Ca. (1982).

Kellogg1983. C.H. Kellogg, "Intelligent Assistants fro Knowledge and Information Resources Management," in *Proceedings of International Joint Conference on Artificial Intelligence*, , Karlsruhe, W. Germany (1983).

Kowalski1979. R. Kowalski, *Logic for Problem Solving*, North Holland (1979).

Lafue1983. G. M. E. Lafue and T. Mitchell, "Data Base Management Systems and Expert Systems for CAD," in *CAD System Frameworks*, ed. K. Bo,North-Holland (1983).

McLeod1982. D. McLeod and R. King, "Semantic Database Model," in *Principles of Database Design*, ed. S.B. Yao,Prentice Hall (1982).

Mylopoulos1980. J. Mylopoulos, P. Bernstein, and H.K.T Wong, "A Language Facility for Designing Interactive Database-Intensive Applications ," *ACM Transactions on Database Systems* **5**(2)(1980).

Parsaye1983. K. Parsaye, "Prolog, Database Management and Knowledge Base Management in Expert Systems Applications," in *Proceedings of ACM/SIGMOD Conference on Modelling of Data*, , San Jose, Ca (1983).

Reiter1983. R. Reiter (Chairman), "A Panel on AI and Databases," in *Proceedings of International Joint Conference on Artificial Intelligence*, , Karlsruhe, W. Germany (1983).

Schoen1983. E. Schoen and R. G. Smith, "IMPULSE: A Display Oriented Editor for STROBE," in *Proceedings of American Association for Artificial Intelligence*, , Washington D.C. (1983).

Smith1981. J.M. Smith, S. Fox, and T. Landers, "Reference Manual for Adaplex," Computer Corporation of America (Jan. 1981).

Smith1977. J.M. Smith and D.C. Smith, "Database Abstractions: Aggregation and Generalization," *ACM Transactions on Database Systems* 2(2) (June 1977).

Smith1983b. R.G. Smith, "STROBE: Support for Structured Object Knowledge Representation," in *Proceedings of International Joint Conference on Artificial Intelligence*, , Karlsruhe W. Germany (1983).

Smith1983. R.G. Smith and J.D. Baker, "The Dipmeter Advisor System. A Case Study in Commercial Expert System Development," in *Proceedings of International Joint Conference on Artificial Intelligence*, , Karlsruhe W. Germany (1983).

Stonebraker1980. M. Stonebraker, "Retrospection on a Database System," *ACM Transactions on Database Systems* 5(2) (June 1980).

Stonebraker1976. M. Stonebraker, E. Wong, and P. Kreps, "The Design and Implementation of INGRES," *ACM Transactions on Database Systems* 1(3) (Sept. 1976).

Vassiliou1983. Y. Vassiliou, J. Clifford, and M. Jarke, "How Does an Expert System Get its Data?," New York University, Center for Research on Information Systems (March 1983).

Zaumen1983. W. T. Zaumen, "Computer Assisted Circuit Evaluation in Prolog for VLSI," in *Proceedings of ACM/SIGMOD Conference on Modelling of Data*, , San Jose, Ca (1983).

IMPLEMENTATION OF RULES IN RELATIONAL DATA BASE SYSTEMS

by
Michael Stonebraker, John Woodfill and Erika Andersen

Dept of Electrical Engineering and Computer Science
University of California
Berkeley, Ca.

ABSTRACT

This paper contains a proposed implementation of a rules system in a relational data base system. Such a rules system can provide data base services including integrity control, protection, alerters, triggers, and view processing. Moreover, it can be used for user specified rules. The proposed implementation makes efficient use of an abstract data type facility by introducing new data types which assist with rule specification and enforcement.

I INTRODUCTION

Rules systems have been used extensively in Artificial Intelligence applications and are a central theme in most expert systems such as Mycin [SHOR76] and Prospector [DUDA78]. In this environment knowledge is represented as rules, typically in a first order logic representation. Hence, the data base for an expert system consists of a collection of logic formulas. The role of the data manager is to discover what rules are applicable at a given time and then to apply them. Stated differently, the data manager is largely an inference engine.

On the other hand, data base management systems have tended to represent all knowledge as pure data. The data manager is largely a collection of heuristic search procedures for finding qualifying data. Representation of first order logic statements and inference on data in the data base are rarely attempted in production data base management systems.

The purpose of this paper is to make a modest step in the direction of supporting logic statements in a data base management system. One could make this step by simply adding an inference engine to a general purpose DBMS. However, this would entail a large amount of code with no practical interaction with the current search code of a data base system. As a result, the DBMS would get much larger and would contain two essentially non overlapping subsystems. On the other hand, we strive for an implementation which integrates rules into DBMS facilities so that current search logic can be employed to control the activation of rules.

The rules system that we plan to implement is a variant of the proposal in [STON82], which was capable of expressing integrity constraints, views and protection as well as simple triggers and alarms for the relational DBMS INGRES [STON76]. Rules are of the form:

```
on    condition
then  action
```

The conditions which were specified include:

the type of command being executed (e.g. replace, append)

the relation affected (e.g. employee, dept)
the user issuing the command
the time of day
the day of week
the fields being updated (e.g. salary)
the fields specified in the qualification
the qualification present in the user command

The actions which we proposed included:

sending a message to a user
aborting the command
executing the command
modifying the command by adding qualification or
  changing the relation names or field names

Unfortunately, these conditions and actions often affect the command which the user submitted. As such, they appear to require code that manipulates the syntax and semantics of relational commands. This string processing code appears to be complex and has little function in common with other data base facilities. In this paper we make use of two novel constructs which make implementing rules a modest undertaking. These are:

1) the notion of executing the data
and
2) a sequence of QUEL commands as a data type for a relational data base system

The remainder of this paper is organized as follows. In Section II we indicate the new data types which must be implemented and the operations required for them. Then in Section III we discuss the structural extensions to a relational data base system that will support rules execution. Lastly, Section IV and V contains some examples and our conclusions.

II RULES AS ABSTRACT DATA TYPES

Using current INGRES facilities [FOGG82, ONG82, STON82a] new data types for columns of a relation can be defined and operators on these new types specified. We use this facility to define several new types of columns and their associated operators in this section.

The first data type is a QUEL command, e.g.

```
range of e is employee
replace e(salary = 1.1*e.salary) where e.name = "John"
```

The abstract data type facility supports an external representation such as that above for a given data type. Moreover, when an object of the given type is stored in the data base it is converted to an internal representation. QUEL commands are converted by the INGRES parser to a parse tree representation such as the one noted in Figure 1 for the qualification "where 13. + employee.salary = 100". Consequently, a natural internal form for an object of type QUEL is a parse tree. Each node in this parse tree contains a value (e.g. 13.) and a type (e.g. floating point constant).

The second new data type which will be useful is an ATTRIBUTE-FUNCTION. This is a notion in the QUEL grammar and stands for anything that can be evaluated to a constant or the name of a column. Examples of attribute functions include:

13.

```
                 RETBOOL
                    |
           +-------F4EQ------------------+
           |                             |
  +---F4PLUS---+                       100.
  |            |
 13.         F4VAR
               |
               1
```

The Parse Tree for the Qualification
Where 13. + employee.salary = 100

Figure 1

1.1*employee.salary +20

newsal

The external representation is the same string format used for objects of type QUEL; the internal representation is that of a parse tree.

Two other data types of lesser significance are also needed, a TIME data type to contain a time of day value and a COMMAND data type to contain a value which is one of the QUEL commands.

Current built-in INGRES operators (e.g. *, /, +, etc.) must be extended for use with attribute functions. In addition, two new operators are also required. First, we need a function new() which will operate with integer data types. When called, it will return a new unique identifier which has not been previously used. Second, we require a partial match operator, ^, which will operate on a variety of data types and provide either equality match or match the value "*".

III INGRES CHANGES

We expect to create two rules relation, RULES1 and RULES2, with the following fields:

```
create RULES1(
    rule-id = i4,
    user-id = c10,
    time    = time,
    command = command,
    relation = c12,
    terminal = c2,
    action = quel)

create RULES2 (
    rule-id = i4,
    type = c10,
    att-fn1 = attribute-function
    operator = c5,
    att-fn2 = attribute-function)
```

For example, we might wish a rule that would add a record to an audit trail whenever the user "Mike" updated the employee relation. This requires a row in RULES1 specified as follows:

```
append to RULES1(
    rule-id = new(),
    user-id = "Mike",
    command = "replace",
    relation = "employee",
    action = QUEL command to perform audit)
```

If additionally we wished to perform the audit action only when Mike updated the employee relation with a command containing the clause "where employee.name = "Fred"" we would add an additional tuple to RULES2 as follows:

```
append to RULES2(
    rule-id = the one assigned in RULES1
    type    = "where"
    att-fn1 = "employee.name"
    operator = "="
    att.fn2 = "Fred")
```

We also require the possibility of executing data in the data base. We propose the following syntax:

```
range of r is relation
execute (r.field) where r.qualification
```

In this case the value of r.field must be an executable QUEL command and thereby of data type QUEL. To execute the rule that was just appended to R1 we could type:

```
range of r is R1
execute (r.action) where r.user-id = "Mike" and
             r.command = "replace" and
             r.relation = "employee"
```

When a QUEL command is entered by a user, it is parsed into an internal parse tree format and stored in a temporary data structure. We expect to change that data structure to be the following two main memory relations:

```
create QUERY1(
    user-id = c10,
    command = command,
    relation = c12,
    time    = time,
    terminal= c2)

create QUERY2(
    clause-id = i4,
    type     = c10,
    att-fn1  = attribute-function,
    operator = c5,
    att-fn2  = attribute-function)
```

If the user types the query:

```
range of e is employee
retrieve (e.salary)
     where (e.name = "Mike" or e.name = "Sally")
       and e.salary > 30000
```

then INGRES will build QUERY1 to contain a single tuple with values:

QUERY1				
user-id	command	relation	time	terminal
current-user	retrieve	employee	current-time	current-terminal

QUERY2 will have four tuples as follows:

QUERY2				
clause-id	type	att-fn1	operator	att-fn2
id-x	target-list	employee.salary	=	employee.salary
id-y	where	employee.name	=	Mike
id-y	where	employee.name	=	Sally
id-z	where	employee.salary	>	30000

Notice that QUERY1 and QUERY2 contain a relational representation of the parse tree corresponding to the incoming query from the user. The where clause of the query is stored in conjunctive normal form, so that atomic formulae which are part of a disjunction have the same clause-id, while the atomic formulae and disjunctions in the conjunction have different clause-ids.

Then we execute the QUEL commands in Figure 2 to identify and execute the rules which are appropriate to the incoming command. These commands are performed by the normal INGRES search logic. Activating the rules system simply means running these commands prior to executing the user submitted command. After running the commands of Figure 2, the query is converted back to a parse tree representation and executed. Notice that the action part of a rule can update QUERY1 and QUERY2; hence modification of the user command is easily accomplished. The examples in the next section illustrate several uses for this feature:

```
range of r1 is RULES1
range of r2 is RULES2
range of q1 is QUERY1
range of q2 is QUERY2
retrieve into TEMP(r1.id, r1.quel) where
          r1.user-id ^ q1.user-id and
          r1.command ^ q1.command and
          r1.time    ^ q1.time    and
          r1.terminal^ q1.terminal

range of t is TEMP
execute (t.quel) where t.id < 0 or
        (t.id = r2.rule-id and
        set(r2.all-but-rule-id by r2.rule-id)
      = set(r2.all-but-clause-id by r2.rule-id
       where r2.all-but-rule-id ^ q2.all-but-clause-id))
```

Rule Activation in QUEL
Figure 2.

The set functions are as defined in [HELD75]. The conditions for activating a rule are:

(i) its tuple in RULES1 matches the tuple in QUERY1 and either

(ii) each tuple for the rule in RULES2 matches a tuple in QUERY2.

or

(iii) there are no required matches in RULES2
(represented by rule-id < 0).

The second condition provides appropriate rule activation when both the user query and the rule do not contain the boolean operator OR. However, a rule which should be activated when two clauses A and B are true will have two tuples in RULES2. This rule will be activated by a user query containing clauses matching A and B connected by any boolean operator. Under study is a more sophisticated activation system which will avoid this drawback.

The commands in Figure 2 cannot be executed directly because set functions have never been implemented in INGRES. Hence, we turn now to a proposed implementation of these functions.

Suppose we define a new operator "|" to be bitwise OR, and "bitor()" to be an aggregate function which bitwise ORs all qualifying fields. Then if we add the attribute "mask" to RULES2, and give each tuple for a particular rule a unique bit, the following query is correct:

```
range of t is TEMP
execute (t.quel) where t.id < 0 or
        (t.id = r2.rule-id and
      bitor(r2.mask by r2.rule-id)
      = bitor(r2.mask by r2.rule-id
         where r2.all-but-rule-id ^ q2.all-but-clause-id))
```

This solution will be quite slow, since the test for each rule involves processing a complicated aggregate. A more efficient solution involves generating masks for all rules in parallel and writing special search code as follows:

```
range of r1 is RULES1
range of r2 is RULES2
range of q1 is QUERY1
range of q2 is QUERY2
retrieve into TEMP(r1.id, r1.quel, mask = 0)  where
          r1.user-id ^ q1.user-id and
          r1.command ^ q1.command and
          r1.time    ^ q1.time    and
          r1.terminal^ q1.terminal

range of t is TEMP

foreach q2 do begin
   replace t (mask = t.mask | r2.mask)
            where t.id = r2.rule-id and
            r2.all-but-rule-id ^ q2.all-but-clause-id
end foreach

execute (t.quel) where t.id < 0 or
                (t.id = r2.rule-id and
            bitor(r2.mask by r2.rule-id)
                = t.mask)
```

Since the value of "bitor(r2.mask by r2.ruleid)" remains constant, the performance of this algorithm can be further improved by including the value of "bitor(r2.mask by r2.ruleid)" in RULES1 and copying it into TEMP as the "acceptmask". The third query would then become:

```
    execute (t.quel) where t.id = r2.rule-id and
                t.acceptmask = t.mask
```

Notice the case where there are no tuples in RULES2 for a particular rule is handled with an acceptmask of zero.

Either a variable length abstract data type "bitstring" or a four byte integer can be used to store the mask. The abstract data type solution has the advantage of allowing an unlimited number of conditions for specifying rule activation, while the four byte integer solution has the advantage of simplicity and speed, but can only represent 32 conditions.

IV EXAMPLES

We give a few examples of the utility of the above constructs in this section. First, we can store a command in the data base as follows:

```
    append to storedqueries (id = 6,
                quel = "range of e is employee
                     retrieve (e.salary)
                     where e.name = "John"")
```

We can execute the stored command by:

```
    range of s is storedqueries
    execute (s.quel) where s.id = 6
```

The following two examples will pertain to the query:

```
    range of e is employee
    replace e(salary = salary*1.5) where e.name = "Erika"
```

To represent this query INGRES will append the following tuples to the QUERY1 and QUERY2 relations:

QUERY1				
user-id	command	relation	time	terminal
current-user	replace	employee	current-time	current-terminal

QUERY2				
clause-id	type	att-fn1	operator	att-fn2
id-z	target-list	employee.salary	=	employee.salary*1.5
id-x	where	employee.name	=	Erika

Suppose we want to implement the integrity contraint to insure that employee salaries never exceed $30,000. Using query modification [STON75] we would add the clause "and employee.salary*1.5 <= 30000". to the user's qualification with the following rule:

```
    append to RULES1(
        rule-id  =  new(),  (call it id-y)
        user-id  =  *,     (matches any user-id)
        command  =  "replace",
        relation =  "employee",
        action   =  "range of Q2 is QUERY2
                    append to QUERY2(
                     clause-id = id-x,
                          type     = "where",
                          att-fn1  = Q2.att-fn2,
                          operator  = "<=",
                    att-fn2   = "30000")
                where Q2.att-fn1 = "employee.salary")"
append to RULES2(
    rule-id  = id-y,
    type     = "target-list",
    att-fn1  = "employee.salary",
    operator = "=",
    att-fn2  = *)
```

Consider a transition integrity constraint that specifies that the maximum salary increase is 20%. This means that the new salary divided by the old salary must be less than or equal to 1.2. This can be achieved by appending a single tuple to R1:

```
append to RULES1(
    rule-id  = new(),
    user-id  = *,
    command  = "replace",
    relation = "employee",
    action   = "range of Q2 is QUERY2
                append to QUERY2(
                 clause-id = id-x,
                      type     = "where",
                      att-fn1  = Q2.att-fn2/Q2.att-fn1,
                      operator = "<="
                      att-fn2  = "1.2")
             where Q2.att-fn1 = "employee.salary""
```

As a last example of an integrity constraint, consider a referential constraint that a new employee must be assigned to an existing department. Such a rule would be applied, for example, to the following query:

```
append to employee (name="Chris", dept = "Toy", mgr = "Ellen")
```

The corresponding tuples in QUERY2 would look like:

QUERY2				
clause-id	type	att-fn1	operator	att-fn2
id-z	target-list	employee.name	=	Chris
id-z	target-list	employee.dept	=	Toy
id-z	target-list	employee.mgr	=	Ellen

Implementation of the constraint requires checking that the department given in the target list of the append appears in the department relation. This is accomplished with the following rule:

```
append to RULES1(
    rule-id  = new(),
    user-id  = *,
    command  = "append",
    relation = "employee",
    action   = "range of Q2 is QUERY2
               append to QUERY2(
                     clause-id = id-z,
                     type      = "where",
                     att-fn1   = "dept.name",
                     operator  = "=",
                     att-fn2   = Q2.att-fn2)
```

```
                    where Q2.att-fn1 = employee.dept"
```

Lastly, protection is achieved primarily by making use of the RULE1 relation, which pertains to the query "bookkeeping" information. Suppose we wanted to ensure that no one could access the employee relation after- hours (after 5PM and before 8AM). The following tuple would be added to the R1 relation:

```
  append to RULES1(
      rule-id  = new(),
      user-id  = *,
      time     = "17:01 - 7:59",
      command  = *,
      relation = "employee",
      terminal = *,
      action   = "range of Q1 is QUERY1
              range of Q2 is QUERY2
              delete Q1
              delete Q2
```

If the query meets the conditions, the action removes the tuples in QUERY1 and QUERY2 and thereby aborts the command.

V CONCLUSIONS

This paper has presented an initial sketch of a rules system that can be embedded in a Relational DBMS. There are two potentially very powerful features to our proposal. First, it can provide a comprehensive trigger and alerter system. Real time data base applications, especially those associated with sensor data acquisition, need such a facility. Second, it provides stored DBMS commands and the possibility of parallel execution of triggered actions. In a multiprocessor environment such parallelism can be exploited.

There are also several deficiencies to the current proposal, including:

a) Rule specfication is extremely complex. This could be avoided by a language processor which accepted a friendlier syntax and translated it into the one in this paper.

b) The result of the execution of a collection of rules can depend on the order in which they are activated. This is unsettling in a relational environment.

c) Rules trigger on syntax alone. For example, if we want a rule that becomes activated whenever John's employee record is affected, we trigger on any query having "employee.name = John" in the where clause. However if the incoming query is to update all employees' salaries, this rule would not be triggered.

d) Commands with multiple range variables over the same relation, so called reflexive joins, are not correctly processed by the rules engine.

e) Aggregate functions have not yet been considered.

f) As noted earlier, boolean OR is not treated correctly.

We are attempting to resolve these difficulties with further work.

REFERENCES

[DUDA78]	Duda, R. et. al., "Development of the Prospector Consultation System for Mineral Exploration," SRI International, October 1978.
[FOGG82]	Fogg, D., "Implementation of Domain Abstraction in the Relational Database System, INGRES", Masters Report, EECS Dept, University of California, Berkeley, CA Sept. 1982.
[HELD75]	Held, G., et. al., "INGRES: A Relational Data Base System," Proc. 1975 NCC, Anaheim, Ca., May 1975.
[ONG82]	Ong, J., "The Design and Implementation of Abstract Data Types in the Relational Database System, INGRES," Masters Report, EECS Dept, University of California, Berkeley, CA Sept. 1982.
[SHOR76]	Shortliffe, E., "Computer Based Medical Consultations: MYCIN," Elsevier, New York, 1976.
[STON75]	Stonebraker, M., "Implementation of Integrity Constraints and Views by Query Modification," Proc. 1975 ACM-SIGMOD Conference on Management of Data, San Jose, Ca., June 1975.
[STON76]	Stonebraker, M. et al., "The Design and Implementation of INGRES," TODS 2, 3, September 1976.
[STON82]	Stonebraker, M., et. al., "A Rules System for a Relational Data Base System," Proc. 2nd International Conference on Databases, Jerusalem, Israel, June 1982.
[STON82a]	Stonebraker, M., "Extending a DBMS with Added Semantic Knowledge," Proc. NSF Workshop on Data Semantics, Intervale N.H., May 1982. (to appear in Springer-Verlag book edited by M. Brodie)
[STON83]	Stonebraker, M., et. al., "Document Processing in a Relational Data Base System," ACM TOOIS, April 1983.

APPLICATIONS OF ARTIFICIAL INTELLIGENCE IN THE KNOWLEDGE-BASED MANAGEMENT SYSTEMS PROJECT

Gio Wiederhold

Jack Milton†

Stanford University

Daniel Sagalowicz

SRI International

Introduction

The Knowledge-Based Management Systems (KBMS) Project* is a research program largely funded by DARPA to investigate innovative approaches to improving the management of large databases. Rather than attempt to build complete systems, we have concentrated our attention on promising areas of research in which we can develop new techniques and prepare effective demonstrations.

We hope that demonstrating the effectiveness of our approaches will cause industry to furter develop these techniques. We believe that the university environment is not well suited for building prototypes of complete systems. We look forward to having database systems in modular form, allowing some of our tests to be performed in a more realistic setting. We describe here only a homogeneous subset of the projects that have been carried out [Wiederhold 83b] since the program's inception in 1977.

Knowledge About Data

Underlying the notion of presenting knowledge about data in a formal manner is the realization that a database and its semantics have a far greater longevity than do any of the programs using that database. The project focuses upon utilization of the semantics inherent in the data - with the objective of representing that semantics accurately and utilizing it to make the database system more effective. Increased effectiveness means 1) making it easier for nonexperts to use and 2) attaining a high system responsiveness and performance. We cannot afford the luxury of having the semantics of a database known only to the

*This research has been supported mainly by the Defense Advanced Research Projects Agency under Contracts MDA903-77-C-322, N39-80-G-132 and N39-82-C-250. Work on the RX project was funded by the National Library of Medicine and the National Center for Health Services Research. Work in natural-language interaction has been partially supported by NSF. Computer facilities were partially provided by the Division of Research Resources of NIH. The views and conclusions contained in this document are those of the authors and should not be interpreted as representative of the official policies, either expressed or implied, of any agency of the U.S. government.

†Permanent affiliation: Department of Mathematics, University of California, Davis.

programmers and contained only within the code of the programs; the knowledge must be represented so it can be properly disseminated and manipulated as the need arises.

These requirements will vary over the lifetime of the database. In the initial phases of its existence usage will consist mainly of data collection and the creation of simple reports. Many database applications do not go beyond this stage. Database managers often consider the successful production of regular reports for subsequent analysis by middle management as sufficient justification for existence of the database.

However, once a large data collection exists, it is a valuable repository of extensional knowledge. This knowledge can be accessed effectively only if there are tools that enable domain experts to use the system without requiring detailed intensional knowledge of the database structure. Database utilization must contend with a formidable obstacle if its only results appear in the form of reports that are scanned by intermediaries. Yet a database structure that is designed to be optimal during initial use will probably be inadequate for the later stage, when enough data will have been collected to make the database genuinely useful for information retrieval and planning.

To partition the problem into subproblems that can be dealt with profitably, we distinguish various types of semantics. Our principal distinction is that of domain versus structural semantics. We expect structural semantics to be largely invariant, whereas domain semantics can change drastically and, in fact, can be modified by users as the need arises.

Structural Semantics

Structural semantics is the relatively stable semantics of the database, typically the descriptions in the schema, and extensions based on other knowledge about the data which may be employed in database design. This knowledge, because of its permanence, can also be used to determine the physical structure of the database.

Structural semantics is related to the functional, multivalued, extension, and inclusion dependencies now employed in database theory [Armstrong 80], [Casanova 82], [Ullman 82]. However, our structural semantics has a somewhat broader scope, since we include also rules for dynamic constraint maintenance in the definition. These permit actions to be specified when requested update and delete operations might violate the database constraints; whereas dependency theory concentrates on the definition of correct states before and after transactions. Thus, structural semantics provides guidelines to the actions carried out by the program that executes those transactions.

We observe that transactions which manipulate many database elements are more frequent and also more important than simple queries. Observe that in order to obtain information from a database many data are collected, aggregated, and used to draw inferences. In order to understand such computations we must go beyond the retrieval of simple facts and typically involve significant subsets of the data. This observation strengthens our concern with performance criteria for databases. We hence require from the model that it permits us to collect knowledge which will help in the physical design of the database. This means that the model should capture much of relevant logical knowledge, but permit transformations so that many implementation alternatives can found and an optimal one can be chosen. Note that, in order to do physical design, we need a model that is independent of physical notions.

The primitive connections of the structural model used to define this semantics are the ownership (1:M), the reference (N:1), the subset (1:1 partial), and identity (1:1 complete). The definition of reference constrains insertion of references to requiring a referent, and deletion of the referent while there exist (N > 0) references. The definition of ownership also constrains the insertion of members, but forces deletion of all the M members with an owner. More complex relationships (N:M) and other partial relationships are constructed by combining these primitives [Wiederhold 79,83a]. Structural semantics is used initially in database design; where it can help specify the transaction programs as well as the data structures to be used for reliable implementation of the semantics.

If a database implementation is known to maintain the structural semantics, this semantics can be profitably exploited. Such integrity may be based directly on the implementation structure, or may be based on constraint rules or procedures attached to the database. For instance, most hierarchical structures enforce the ownership connection. If the database system is inadequate, and all of our systems are lacking some of the required constraint maintenanace capabilities, then it falls on the programmer to enforce the constraints. The structural model is then used as a formal specification method.

Today progammers implement databases and their application semantics in an *ad hoc* way. Since people re smart, they are often correct, but some transactions may permit erroneous actions: Will all `inventory` entries have a correct current `supplier` reference? Will they all have `part+type` numbers appearing in the approved list? If that can be guaranteed, through the datastructure if possible, then the `inventory` total will be equal for either a `supplier`-based query or a `part+type`-based query. Few databases, other than those used in examples to `find the manager of the toy department`, will be able to pass that filter.

The limited knowledge contained in traditional databases is stored in the schema. Building a knowledge-based front-end to enhance traditional databases means expanding the capability of the schema. Alternatively, when the schema is not easily changed (as is often the case), it means building what amounts to a coprocessor for the database schema program. We look forward to having database management systems which are either expandable in that sense, or which are composed of replaceable modules.

Structural semantics can be used specifically for the following purposes:

1. Update verification. The formal constraints of the model provide a basis for verifying that the current update does not violate the specified interrelational dependencies. Use of a general model rather than distinct rules helps in assuring completeness of coverage. Example: A statement to insert a new inventory record can avoid the explicit subquery ... `WHERE part.supplier+name = supplier.s+name`.

2. Simplified query processing. This is made possible by using the knowledge embodied in the structural model. Since the connections among relations are known, it is not necessary for the user queries to specify all the joins to be performed. Only if multiple connections exist between query objects is path verification necessary. Even in such cases only the connecting attributes have to be named explicitly [ElMasri 81]. Example: `Find weight of part X` can be processed, although the `weight` of a `part` is actually given in the referenced `part+type` entry.

2. Simplified transaction processing. If the structure of the database is automatically maintained by reference to the model, the transaction programs will not have to check for structural error conditions. Retrieval transaction can be optimized and be assured of identical results. Example: `Find Total(value)`

of inventory can become Find Total(Total(value) of inventory by part+type, which may be more efficient of good access paths for part+types exist.

To make proper use of structural semantics, we have to guarantee that all updates will satisfy the constraints that have been defined [Keller 81]. Maintaining the integrity of the database requires that updates be checked carefully. The validation of updates, however, is viewed as satisfying the constraints not only of the model, but, more importantly, of the underlying semantics of the database as well – so that users will not have to contend with an inconsistent database during subsequent query sessions. We have shown that use of a model, triggered by attempted changes to attributes involved in connections, can guarantee that the specified constraints are satisfied. It is well known that the cost of correcting an error during an update when the pertinent documents and the information to be entered are available is but a fraction of the cost of correcting the database days or even years later when retrieval results fall behind expectations.

The information imbedded in the structural model can also indicate when a query has been asked inappropriately. An example of a query with an implied but false assumption is:

How many widgets were delivered to the receiving department in New York yesterday?

The formal equivalent of this query becomes:

```
SELECT deliveries.number WHERE
                deliveries.part+type = 'Widget' AND
                deliveries date = 10CT1983 AND
                deliveries.dep+number = departments.number AND
                departments.name = 'Receiving' AND
                departments.location = 'New York'.
```

The system's response will be erroneously none – because the receiving department is actually in New Jersey.

This question does not fail at the level of accessing widgets, but at the level of accessing the list of departments. Feedback about such a failure (Requested department not found.) can help the user rephrase the queries and not come away with erroneous information [Kaplan 81].

We do not believe that the formal queries are convenient for the users when many relations are involved. This means that implied assumptions can enter formal queries as well as the more concise knowledge-based ntural language queries.

Views

The problem of handling database updates is complicated by the observation that the updating user has only a limited view of the entire database and hence cannot be expected to understand all the ramifications of an update. In today's practice most database updates are being performed by clerical personnel handling programmed transactions. Although the latter are programmed to preserve the consistency of the database, they typically lack an underlying model. Here too, our research here employs two approaches: algorithms and the application of artificial intelligence techniques.

In the algorithmic approach, we are developing methods to minimize update ambiguity and to enumerate the remaining database states that can satisfy an update request [Keller 82]. In static views the alternatives, not all of them obvious, can be presented to a database administrator for selection.

In dealing with updates in natural language, the user's view must be assumed to be restricted to the information he has previously obtained from the database. Given that limited view, the ambiguity increases and heuristic techniques must be used to determine the most reasonable update. Here the candidate transformations which can satisfy the updates are ranked based on a minimal effect on structures not in the users' view [Davidson 82].

In both approaches the number of possible alternatives is limited by the constraints of the structural model.

Domain Knowledge

Stable structural knowledge is complemented by domain knowledge. Although there are many types of domain knowledge, we shall not undertake to classify them here but shall limit ourselves to some useful examples. Domain knowledge can vary over time and can be changed directly by the users of the database or indirectly through a learning mechanism. Since this type of knowledge can be altered, it cannot be bound to the database structure. In most implementations we would expect domain knowledge to require a higher level of interpretation at query time. Since it is less precise, it can also be more powerful. We do not have to deal with that knowledge in the same rigorous fashion needed for structural knowledge.

An example is the knowlege that 'oil is carried in tankers'. This statement is true only to a certain level, like a frequently used syllogistic example in artificial intelligence postulating that "birds fly". These statements disregard logically important, but quantitatively minor facts, as the oil carried in barrels on freighters and the existence of penguins. The utilization of this deliberately restricted knowledge, however, can be extremely effective. A query requesting `the amount of oil imported by the United States in 1982` can thus be processed by simply scanning through the indexed list of `tankers` and noting whether they used U.S. ports during the year. The precise answer to the query on the other hand, would necessitate checking through every bill of lading for all ships to determine what they carried [King 81].

Of course, domain knowledge can also be used to reduce the number of alternatives for updating. The notion that employees are more easily reassigned than departments, for example, reduces the number of alternatives when an employee is assigned to a new location.

Focus Knowledge

A form of domain knowledge, which is triggered by the current activity on the database, rather than by the content of the database, can be used to improve the focus of the presentation of data. We use focus knowledge, for instance, to suggest related data to the user. When the focus is `transportation` then a query to the inventory will retrieve all attributes in the query path which are of type `location` or `time`. The same query, within a focus of `purchasing` will include all attributes of type `money`. Our suggestions have been been based on explicit task models, but one might also build task models by tracking the activities of experts.

If we know the type of task involved, the suggestion is implemented by letting the screen display all relevant information in response to a query. For instance, in a naval rescue mission, not only should the obvious data on the lifeboats and medical supplies be presented, but also the time and fuel available to reach the rescue point.

An Application of Knowledge to Data Analysis

Associated with tasks may be common analytic procedures. Knowledge about procedures is needed to perform the appropriate data analysis. The expert who knows the procedures is rarely present when a user explores the database. Rules can be used to select statistical methods appropriate to the data. Furthermore, knowledge about the application can serve as a basis for including all known relationships among the data in an analysis model.

Content analysis requires large collections of well maintained data. We have available to us a large database of rheumatolgy patients. Many years of operational use were required to get to this stage. In this research we have developed tools that use artificial intelligence techniques to assist in the processing of time-oriented data. A frame-based model is used to capture knowledge about the medical domain. This model can assist in combing the database for unexpected time-lagged correlations and report such events as prospective hypotheses. If analysis of the finding appears warranted, a statistical model is built that includes all possible covariates given in the knowledge base.

The appropriate statistical tests are selected by applying a collection of rules that use the descriptions of the data from the database schema and the conditions that are satisfied by the statistical procedures. If the findings turn out to be significant, they are added to the knowledge base for use in another iteration of hypotheses generation and testing. This approach to knowledge generation complements the familiar but less formal approach taken by expert systems [Shortliffe 73,79].

This approach has been demonstrated by using a small subset of the time-oriented rheumatology database (ARAMIS) [Fries 79]. The statistical methods now available are limited to multi-variate regression. Tentative hypotheses were generated and automatically tested for statistical significance. Several verified hypotheses could be added to the original loaded knowledge base [Blum 82].

Summary

This brief summary on the use of semantic knowledge in database systems should illustrate the great variety and depth of problems we can seek solutions to by combining knowledge and data. We always make the assumption that, while our database can be extremely large and change rapidly, the underlying semantics we find so useful is relatively much smaller and changes only gradually. By achieving adequate coupling of the database with intensional knowledge, however, we can help make the database more accessible to planners and decision-makers [Davidson 1980].

In principle, the major objective of this work is to codify the knowledge that is now distributed among the many programs used to access most databases. Since we expect the databases to outlive both the programs and the availablility of the programmers who did the original codification, it is essential that we move up to new levels of knowledge management if we want to be able to handle the valuable data resources we acquire.

References

[Armstrong 80]
W.W. Armstrong and C. Delobel: "Decompositions and Functional Dependencies in Relations"; ACM TODS, Vol.5 No.4, December, 1980, pp.404–430.

[Blum 82b]
R.L. Blum: *Discovery and Representation of Causal Relationships from a Large Time-Oriented Clinical Database: The RX Project*; Lecture Notes in Medical Informatics, No.19, Lindberg and Reichertz, (eds.), Springer-Verlag, New York, New York, 1982, 242 pp.

[Casanova 82]
M.A. Casanova, R. Fagin, and C.H. Papadimitriou, *Inclusion Dependencies and Their Interaction with Functional Dependencies*; IBM Research Report RJ3380 (40160), August 3, 1982.

[Davidson 80]
J. Davidson and S.J. Kaplan, "Parsing in the Absence of a Complete Lexicon"; *Proc. 18th Annual Meeting of the Assoc. for Computational Linguistics*, Philadelphia, Pennsylvania, June 19–22, 1980.

[Davidson 82]
J. Davidson, "Natural Language Access to Databases: User Modeling and Focus"; *Proc. 1982 CSCSI/SCEIO Conf.*, Saskatoon, Saskatchewan, Canada, May 17–19, 1982, pp.204–211.

[ElMasri 81]
R. ElMasri and G. Wiederhold, "Formal Specifications of GORDAS: a High-level Query Language for Graph Databases"; *Proc. 2nd Intl. Conf. on the Entity-Relationship Approach to System Analysis and Design*, Washington, D.C., October 12–14, 1981.

[Fries 79]
J.F. Fries and Dennis McShane, "ARAMIS, A National Chronic Disease Databank System"; *Proc. 3rd Symp. on Computer Applications in Medical Care*, Washington DC, IEEE, October, 1979, pp.798–801.

[Kaplan 81]
S.J. Kaplan and J. Davidson, " Interpreting Natural Language Updates"; *Proc. 19th Annual meeting*, Assoc. for Computational Linguistics, June 1981.

[Keller 81]
A.M. Keller and G. Wiederhold, "Validation of Updates Against the Structural Database Model"; *Symp. on Reliability in Distributed Software and Database Systems*, IEEE, July 1981, Pittsburgh, Pennsylvania, pp. 5–199.

[Keller 82]
A.M. Keller and G. Wiederhold, "Updates to Relational Databases Through Views Involving Joins"; *Proc. 2nd Intl. Conf. on Databases: Improving Usability and Responsiveness*, Jerusalem, Israel: Academic Press, June 1982, pp. 363–384.

[King 81]
J.J. King, " QUIST: A System for Semantic Query Optimization in Relational Databases"; *VLDB 7*, Zaniolo and Delobel (eds), September 1981, pp.510-517.

[Shortliffe 73]
Edward Shortliffe, S.G. Axline, B.G. Buchanan, T.C. Merigan, and S.N. Cohen, " An Artificial Intelligence Program to Advise Physicians Regarding Antimicrobial Therapy"; *Computers and Biomedical Research*, Vol.6, 1973, pp.544–560.

[Shortliffe 79]
E.H. Shortliffe, B.G. Buchanan, and E.A. Feigenbaum, " Knowledge Engineering for Medical Decision Making: A Review of Computer-Based Decision Aids"; *Proc. of the IEEE*, Vol.67, no.9, 1979, pp.1207–1223.

[Ullman 82]
J.D. Ullman, *Principles of Database Systems, 2nd ed.*; Computer Science Press, 1982, 494 pp.

[Wiederhold 79]
G. Wiederhold and R. ElMasri, "The Structural Model for Database Design"; *Proc. Intl. Conf. on Entity-Relationship Approach to Systems Analysis and Design*, North Holland Press, December 1979, pp. 247–267.

[Wiederhold 83a]
G. Wiederhold, *Database Design*; McGraw-Hill, Computer Science Series, 2nd ed., January 1983, 768pp.

[Wiederhold 83b]
G. Wiederhold, J. Milton, and D. Sagalowicz, *The Knowledge-Based Management Systems project*; Stanford University, Stanford, California, June 1983.