Report 85-11
Stanford -- KSL

Scientific DataLink

Graphics for Knowledge Engineers: A Window on Knowledge Base Management.
Shoko Tsuji, Edward H. Shortliffe,
Apr 1985

Graphics for Knowledge Engineers: A Window on Knowledge Base Management

Shoko Tsuji[1]

Edward H. Shortliffe

Medical Computer Science Group

Knowledge Systems Laboratory

Departments of Medicine and Computer Science

Stanford University School of Medicine

Stanford, California 94305 -- (415) 497-6979

MEMO KSL-85-11 -- April 1985

This work has been supported by the National Library of Medicine under Grants LM-03395 and LM-00048, the Office of Naval Research under contract NR 049-479, and the Division of Research Resources under Grant RR-01613. Dr. Shortliffe is a Henry J. Kaiser Family Foundation Faculty Scholar in General Internal Medicine. Computer facilities were provided by the SUMEX-AIM resource under NIH Grant RR-00785.

---

[1]Current address: Teknowledge, Inc., 525 University Avenue, Palo Alto, CA 94301, (415) 327-6600

Table of Contents

List of Figures

Abstract

Optimal construction of expert systems demands a powerful interactive environment for knowledge base management by knowledge engineers. Key requirements include techniques for (a) examining existing information, (b) adding new knowledge and editing preexisting data structures, and (c) examining dynamic internal system behavior to facilitate debugging during consideration of actual cases. This article addresses design considerations in the development of such knowledge engineering tools. The emphasis is on the use of professional workstations with high density graphics interfaces since that technology is rapidly becoming a predominant environment for knowledge engineering activities. To illustrate the points made, examples are drawn from a prototype knowledge base management tool designed for knowledge engineers constructing the knowledge base of ONCOCIN, an expert system to advise in the management of cancer patients.

Introduction

Expert systems research has tended to concentrate more on the development of system building tools than on the construction of knowledge base management tools. These are, of course, interrelated topics since optimal knowledge engineering requires careful attention to issues involved with knowledge base management: examining existing information, updating the knowledge base, adding new information, and debugging the complex interactions of an expert system. Tools to facilitate these tasks can greatly affect the speed with which an expert system can be developed, modified, and expanded to meet users' expectations. Essential tasks become cumbersome without the use of tools to simplify access to, and understanding of, the various knowledge components.

An approach that is being increasingly explored is the application of graphics techniques to the problem of knowledge base management. Tools that use conventional display terminals cannot optimally aid in visualizing the interaction of various components of the knowledge base or in anticipating the consequences of these interactions. In addition, user interaction is restricted to simple command languages or a subset of natural language. In the 1970's, the time-sharing systems which were used for expert system development were unable to supply the dedicated computational resources necessary for interactive graphics. In fact, observers often asked whether these CPU- and memory-intensive expert systems would ever be practical. In the 1980's, however, we are seeing the introduction of powerful single-user workstations that are equipped with high resolution graphic displays and are capable of satisfying an expert system's requirements for high speed and large memory. Graphics can enhance textual explanations with diagrams, present a comprehensive view of system elements as well as detailed descriptions of individual facts, and provide a mechanism for focusing attention on different subcomponents of what is displayed. A graphical interface that uses simple pointing devices can also simplify interaction between the user and the computer while increasing the range of communication.

In this article we discuss considerations in the design of knowledge base management tools for use in the knowledge engineering process. Our emphasis is on the use of professional workstations with high density graphics interfaces since we believe that this technology will rapidly become a predominant environment for knowledge engineering activities. To illustrate our points, we will draw examples from our work building a prototype knowledge base management tool [1] for use by knowledge engineers on the ONCOCIN Project at Stanford

University [2].

Knowledge Engineering and the Management of Large Knowledge Bases

Typical knowledge bases contain synthesized data, empirical judgments, problem solving strategies, and facts about objects. They are the source of power for expert systems, programs that are generally designed to give advice using both formal knowledge and the judgmental expertise of specialists. The construction of such knowledge bases has become known as *knowledge engineering*. Knowledge base management entails keeping the information updated, consistent, and error-free. On the surface, this task may sound straightforward; however, it is complicated by the growing size and complexity of a developing knowledge base. The information inevitably becomes difficult to manage, even for those who originally encoded it and know its organization. In addition, during system development, knowledge engineers find they must often extend or modify the capabilities of the system building tool they are using. As a result, information which should be explicitly represented in the knowledge base may become inadvertently hidden in code. The resulting problems include uncertainty about where to find important information and an inability to enhance or repair the system successfully and rapidly.

Expert systems methodology has matured greatly in recent years, resulting in some formalization of the system building process [3]. Despite these efforts, the field remains one that is primarily taught by apprenticeship and hands-on experience. Much work remains to be done before the general principles of knowledge engineering will be completely formalized. One major shortcoming has been that until recently there had been few attempts to test these systems outside the academic environment. With the emergence of young companies established to build commercial applications of expert systems, we may begin to see formalization efforts based on corporate experiences in building large numbers of systems and interacting with diverse clients.

Expert systems should be viewed as long term projects that continue well beyond the initial development stage. This implies that improved approaches to knowledge base management should be seriously explored, since the speed with which systems can be fielded and modified will affect the eventual acceptance or rejection by intended audiences. Unless such systems can easily grow and change as their users' needs evolve, they are likely to fall

short of their intended goals.

Our work on the ONCOCIN project typifies expert systems research in an academic environment. However, since the prototype system is already in experimental use by physicians, we are beginning to define long range plans for its dissemination outside the university setting. As ONCOCIN's knowledge base has grown, we have begun to anticipate a need for improved knowledge base management as the system is made more widely available. ONCOCIN is a medical consultation system designed to help physicians with the management of cancer patients enrolled in experimental chemotherapy protocols. Project members face a problem common to large, expanding expert systems: without some processing, much of the information that can be retrieved from ONCOCIN's knowledge base is either poorly organized for use by knowledge engineers or unrelated to their needs.

ONCOCIN's knowledge base currently contains three hundred seventeen rules, three hundred seventy five parameters, twenty-two control blocks, and encoded information for forty-two Hodgkin's Disease, non-Hodgkin's lymphoma, oat-cell carcinoma, and breast cancer protocols. Furthermore, the knowledge base is continuing to expand due to incorporation of other treatment protocols. The prototype knowledge engineer's tool we have developed on a professional workstation[2] has forced us to consider some of the general problems involved with knowledge base management and has provided opportunities for solutions using graphical techniques. After briefly describing the kind of workstations for which our analysis and solutions are most relevant, we will describe the key issues in knowledge base management and illustrate the concepts using the ONCOCIN prototype tool.

Personal Workstations

The *personal* or *professional* workstations on which large expert systems are increasingly being implemented share a number of common features: a high resolution graphics display, a keyboard for conventional typed input, a pointing device such as a mouse, high level languages suitable for AI applications (especially LISP), and large memories and address spaces. A color display and audio output are optional with some manufacturers. Such workstations can be connected to one another by a high bandwidth network, thereby allowing sharing of high cost peripherals, such as remote file servers and high quality printers. Each machine typically

---

[2] the Xerox 1108 or *Dandelion*

has from one to five megabytes of main memory and from 30 to 100 megabytes of local disk storage which is used to provide the large virtual memory. They may also come with a large user-writable microstore, which provides the support for graphics and high level languages.

Moving research to personal workstations from traditional large time-shared machines offers several advantages. A high quality interactive user interface, plus the instant availability of enough processing power to update the graphics display, would be impossible to support in a time-shared environment. Workstations offer improved and constant response since they can be dedicated to the task at hand. In addition, the user has access to the full power of the workstation without the loss of any resources that are normally available on a time-sharing system. If one workstation in a local network becomes unavailable, none of the others is affected; if files are stored on a remote file server, users can move to other machines and continue their work. An added benefit is that special experimental hardware, such as is needed for some applications, can be added and it will get the immediate response it needs. Finally, systems developed on these computers can be easily exported if users acquire the same or a comparable machine. Since these machines are already much less expensive than their mainframe counterparts, and unit costs are continuing to drop dramatically (a 50% reduction in one workstation's price in the last two years, for example), it is increasingly reasonable to expect that interested users will be able to obtain the necessary hardware and to apply it in a cost-effective manner. We believe that the trend is clear: although personnel costs are increasing, the price of high performance computers is going down; with continued improvements in hardware, current and future AI systems development will accordingly move to networks of personal workstations.

Knowledge Base Management Subtasks

The knowledge engineering process can be facilitated by several different knowledge management capabilities. These include

- facilities for knowledge base examination
- simplified methods for entering new information and modifying pre-existing knowledge structures
- interactive capabilities to permit system debugging while running test cases

In the following sections we will describe these functions in turn and give examples drawn from ONCOCIN's knowledge base management tool.

Knowledge Base Examination

Generic Issues

Knowledge base browsing systems developed on conventional character-oriented display terminals have a limited scope for presenting information. Such systems have typically depended on text formatting techniques to give the impression of structure. Very simple diagrams can be constructed on such terminals, but they are hard to generate, and generally provide a highly constrained view of the system or knowledge structure. Textual explanations can provide sufficient details about individual components of a knowledge base, but they fail to provide a comprehensive view of the entire knowledge base or even a major subsection.

A graphical display allows greater flexibility in representing information. Human ability to take in information holistically can be exploited. One example is the use of graphs and networks to represent the interactions among data structures or to illustrate control flow in a knowledge system. Hierarchical relationships in either the domain knowledge or rules can be explicitly conveyed. A carefully constructed graph or diagram can convey a great deal of information in a form which is not overwhelming or confusing and would be virtually impossible to convey in text. In addition, the user's interaction through a graphical system can take advantage of the interconnections between data structures, resulting in a smoother transition that simplifies the user's grasp of how sequential elements interrelate. Graphical techniques also allow highlighting of information, e.g., by inverting or underlining text regions and using different styles and sizes of fonts.

Most modern workstations with graphics capabilities also provide a mechanism called *windows* for organizing and segmenting information on the screen. Windows are special regions of the screen that can be treated somewhat like pieces of paper on a desk. The user can create and destroy them at will, rearrange them on the screen, and overlap them on top of each other. The user can choose to monitor or ignore the contents of each window. Window placement can be used to reflect how often information is referenced. Windows can be placed near eye-level (toward the middle to top of the screen) if it is anticipated that they will most frequently be referenced, whereas windows placed near the fringes of the screen (most notably on the bottom) can be easily ignored unless a special need dictates their examination.

The concept of browsing via a graphical display is not new. For example, the Steamer Project has experimented with different graphical representations of data [4], primarily for pedagogical purposes. In addition, a knowledge-based programming system called LOOPS [5] has exploited graphs to facilitate the exploration and debugging of its knowledge bases.

Examples from the ONCOCIN Browsing System

Early versions of ONCOCIN originally had no facility for skimming through the knowledge base. Knowledge engineers had to rely on two sources of information about ONCOCIN's domain and knowledge base: a one hundred and fifty page listing which divided the knowledge base into categories based on datatypes, and a small booklet comprised of *chemotherapy cards* that are used by oncologists when caring for protocol patients. These kinds of knowledge base descriptions can only be utilized after sorting out irrelevant information and following pointers to other structures, a job much more suited for computers than for people. The information in the chemotherapy card is difficult to translate directly to individual elements of the knowledge base; it is meant for a different type of user: physicians.

ONCOCIN's prototype browsing system exploits the interdependencies which exist among the major datatypes in the advice program: contexts, parameters, rules, and control blocks [6]. *Contexts* embody domain concepts such as diseases, protocols, and chemotherapies. They serve as a mechanism to focus ONCOCIN's reasoning process by partitioning the knowledge base during a consultation. *Parameters* are selected attributes of the patient (test results, drug doses, etc.) that ONCOCIN uses to formulate a recommendation. *Rules* are the standard datatype found in many expert systems, such as MYCIN [7]. *Control blocks* are a high level description of a system's method for performing tasks. They provide a separation of control from inferential knowledge.

The browsing system aids ONCOCIN's knowledge engineers by providing an overview of the major datatypes as well as simplifying access to related information. The contexts are represented as hierarchies. A graph represents the relationship between diseases and chemotherapies. Each disease can be treated with a set of mutually exclusive chemotherapies. Information about any of the diseases or chemotherapies on this graph can be retrieved using the mouse pointing device. The buttons on the mouse have different effects when one is pushed while the mouse-controlled arrow on the screen is pointing at a node. This process of using the mouse to point at an element on the screen and then pushing one of the mouse

buttons to select a specific function is informally referred to as *buttoning* an item. For example, if the user points at a chemotherapy node on the screen, using the mouse, the middle button can be used to constrain further information to a particular branch of the graph (Fig. 1). The bold face lines and inverted text in the figure indicate that the knowledge engineer has chosen the situation to be the chemotherapy PAVE in Hodgkin's Disease.

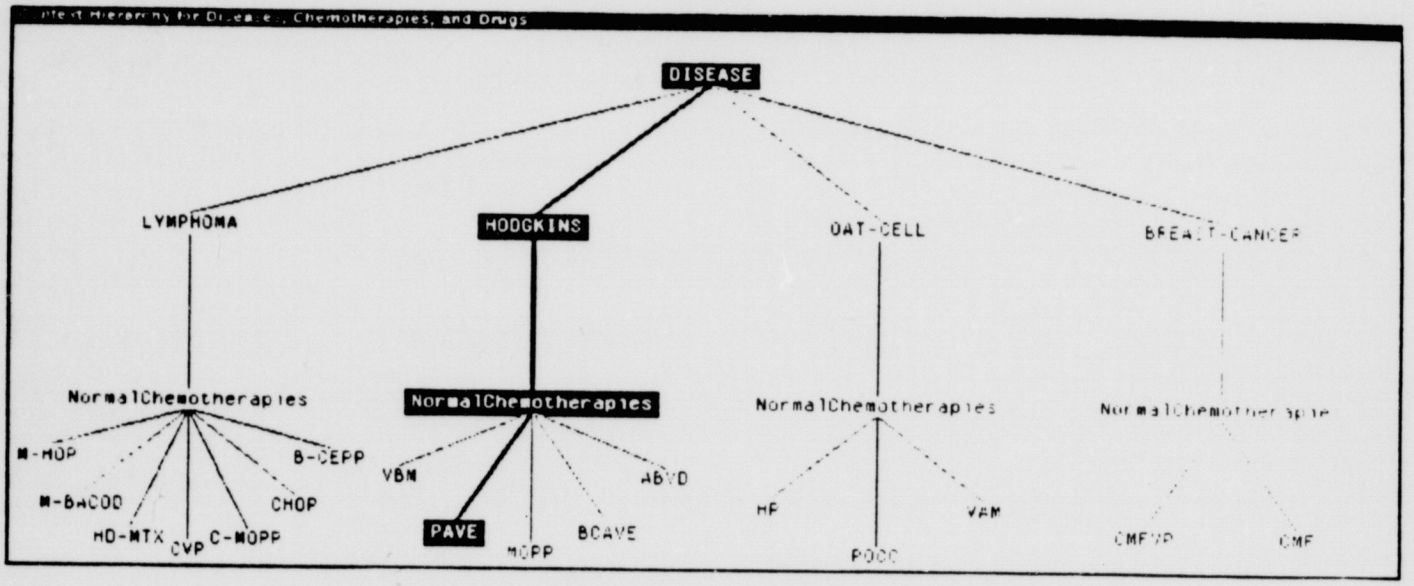

Figure 1: Context Hierarchy - constrained to PAVE

Pushing a mouse button often causes a special menu to appear on the screen in relation to the item at which the user had been pointing. These so called *pop-up* menus can then be referenced (using the mouse) to select specific subfunctions of interest. For example, when pointing at a chemotherapy node, pressing the left mouse button causes a pop-up menu to appear with the choices *General Information, Display a Chemotherapy Card, Inspect Property List,* and *Edit a Chemotherapy.*

Fig. 2 shows the result of selecting *Display a Chemotherapy Card* for PAVE. If instead *General Information* is buttoned, the results appear in two windows; one contains a graphical display (Fig. 3) and the other contains a textual description of PAVE. If further information is desired about any of the protocols or drugs displayed in Fig. 3, the mouse is used to select the item.

The interaction of rules and parameters is especially difficult to visualize and convey

PAVe Protocol Card

PAVe

Procarbazine, 100 mg/m2 PO q.d. days 1 - 14

Alkeran, 7.5 mg/m2 PO days 1, 2, 8, 9

Velban, 6 mg/m2 IV days 1 & 8

WBC		Platelets		Dosage Modification % Calculated Doses Pro	Alk	Vel	Rules
A-cycle PAVe							
>	3.5	>	120	100	100	100	Default
3	3.5	80	120	50	75	75	137 18 34 29
<	3	<	80		Delay		35 36
B-cycle PAVe							
>	3.5	>	120	100	100	100	Default
3	3.5	80	120	50	75	75	137 18 34 29
		50	80	25	50	75	194 165 19
2.5	3			25	50	50	21 20
2	2.5			Omit	25	25	138 27
<	2	<	50		Abort		43 41

Figure 2: PAVE protocol card

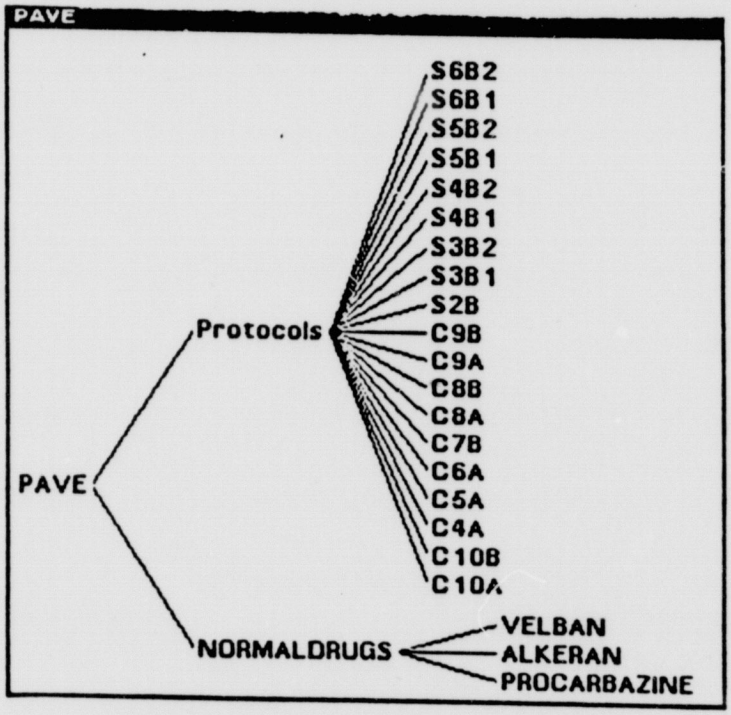

Figure 3: General Information for PAVE

through textual output alone. With graphics, a more natural representation of a network

structure is used, where rules and parameters are the nodes of the network and their relationships are links. A simple interaction might proceed as follows. The knowledge engineer defines a context to focus the reasoning chain, by using the mouse and a series of menus. The window titled *Rule Tree* (Fig. 4) shows the result of examining the parameter Attendose when the context has been defined to be *adriamycin in CHOP for patients with any lymphoma.*[3] To the right of Attendose are the links to the rules which conclude values for Attendose. To the extreme right of this window are the hematological parameters (WBC and Platelets) that are used to determine the amount to attenuate.

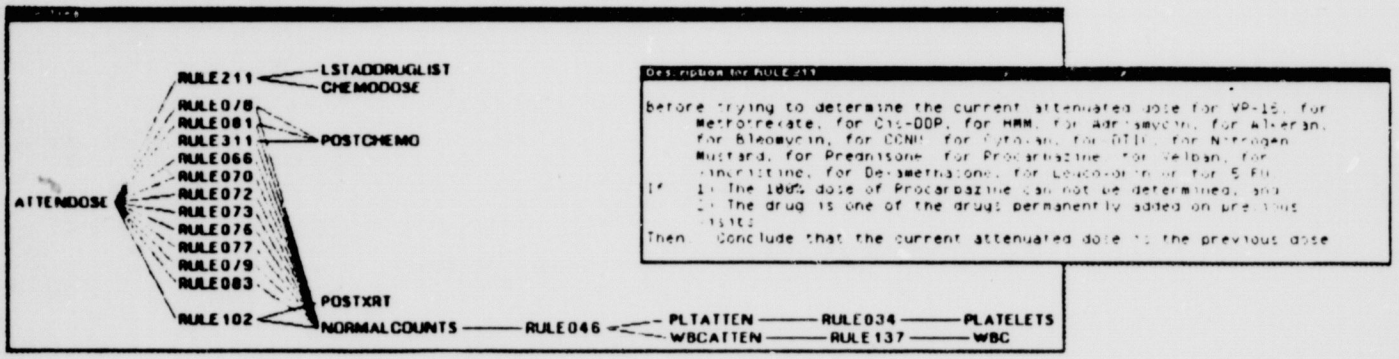

Figure 4: A graph with Attendose as the root of the backward chaining network along with the description for Rule 211.

The translation of any rule in the network may be obtained by buttoning that rule's name on the screen. The knowledge engineer has a choice of either a LISP or English translation. Rule translations appear in a separate window on the screen with the graphical display of the rules remaining for reference. Fig. 4 shows the result of selecting Rule 211 from the graphical display. Buttoning a parameter allows the knowledge engineer to inspect the parameter's property lists. The rules and parameters can also be selected for editing using the middle mouse button.

The knowledge engineer can also choose to examine control blocks through a series of

[3]Attendose is the parameter used by ONCOCIN for determining the extent to which the dosage of a drug should be reduced (attenuated) due to prior toxicity or complications.

menus. Since control blocks represent procedural tasks, they are comprised of discrete steps. The textual translation of the steps is displayed in a window. Steps which specify that another control block should be executed, or that the value of a parameter must be found, can be selected with the mouse. Items that may be selected for further information are indicated to the user with bold-faced text; buttoning other portions of the control block will have no effect. If the selected step invokes a control block, then that control block will also be displayed. If a parameter value must be found, then the reasoning chain will be displayed as a network. (See Fig. 4).

As these examples illustrate, the workstation environment is well suited for the task of examining a large knowledge base. Graphically displayed information is communicated succinctly and rapidly. Such a system allows the juxtaposition of an overview with a detailed description of one of the components. It is much easier to retrieve information using a mouse than a query language. This type of system can also serve as an instructional medium for those people who wish to learn more about how the knowledge base is organized and how the individual elements interrelate.

Entering and Modifying Information

Generic Issues

Most knowledge engineering projects adapt available display or line editors for information entry and modification. The line editors are modified to process templates associated with each data structure, and the knowledge engineer can edit the values associated with each slot in the template one at a time. These editors are also enhanced with capabilities to perform error checking such as spelling correction and value testing, along with necessary bookkeeping chores to keep the knowledge base consistent.

There are several inherent shortcomings with conventional editors. Perhaps most important, knowledge engineers are unable to refer to other components of the knowledge base simultaneously on the terminal screen during editing. Screen space is a scarce resource, and to utilize it for editing often precludes its use for display. The knowledge engineer must therefore generally rely on hard copy listings of the knowledge base. The knowledge base listing is frequently organized by datatype groups in order to facilitate locating items of interest. Despite this ordering, it is time-consuming to locate a specific item and distracting to divide attention between the screen and the listing.

The interaction with a graphically-based editor is much more natural than with conventional editors. User interaction in a graphical system is accomplished using windows, menus, and a mouse rather than a command language and keyboard. Menus are used to display lists of possible commands or available options at a particular point in time. If one uses menus and mouse pointing devices, it is no longer necessary to associate a set of commands with a particular command mode. Instead of using keyboard commands to manipulate what is being displayed, the mouse is used to point directly at graph nodes, diagrams, or regions of text to invoke the editor or to prompt for more information.

As a result, graphical editors are powerful yet easy to learn. The need to memorize key stroke sequences for commands can be replaced through the use of the mouse to select command choices from a menu. This mode of interaction is faster than typing and easier to learn. It is more intuitive to use the mouse to select an item to edit and then select an edit command from a menu.

Examples from ONCOCIN's Editing System

Our prototype knowledge base editor was built using the Interlisp-D structure editor [8] and was then integrated into a line-oriented data structure editor that had been built for ONCOCIN before its conversion to workstations. Consequently, the editor preserves features that are familiar to project members such as borrowing existing property values, parsing the new property values, and separating the editing changes from the current expert system. Isolating the editing changes enables several people to modify and work on different pieces of the knowledge base without interfering with one another. Since the addition of new datatype instances is a simple modification of existing datatypes, the knowledge engineer can extract property values from the desired datatypes using a menu. This menu differs from standard pop-up menus since it allows the knowledge engineer to make more than one selection. This specialized menu was developed because the use of several pop-up menus to make multiple choices was too distracting. The original editor allows values to be edited only one at a time. The workstation editor, like a display editor, allows all property values to be edited during the same operation.

The editor can be easily invoked while examining data structures or during actual debugging without losing any of the context. The knowledge base tool was also constructed to provide immediate feedback on editing changes. The modification is incorporated into the existing knowledge base and, depending on what is currently on the screen, may actually be

reflected in the displayed structures.

To illustrate how a datatype instance is edited from a graph, suppose a user has buttoned (selected) the node representing RULE311 shown in Fig. 4. The information associated with RULE311, such as its condition and conclusion, what parameters refer to the rule, etc., is displayed. The *condition* is edited to add another parameter to be tested. After finishing the change, a message is printed stating that a parsing error was found in the condition. Inspecting the condition, which has reappeared in the edit window, the user sees that the new parameter was misspelled. After this is corrected, the graph is updated.

Large knowledge bases suffer from the problems of redundancy, inconsistency, and omission of data. They need a critiquing facility which can compare and analyze new rules against existing rules in the knowledge base. We earlier developed a prototype rule checker for ONCOCIN [9], but the procedure is exhaustive and, therefore, time-consuming (each rule must be compared against every other rule in the knowledge base). The system has accordingly been modified so that it is integrated into the graphical environment and the user can select subcomponents of the knowledge base for which focused conflict analysis is appropriate. The rule checker's output routines also take advantage of the workstation's graphics capabilities. The addition of system meta-knowledge may someday help to analyze rules intelligently and to reduce the search space to a subset of the knowledge base.

Dynamic System Debugging

Generic Issues

In the workstation environment, sophisticated tools have been developed to handle some of the problems of rapid design upheavals that AI programs generally undergo [10]. Expert systems initially go through a phase of rapid prototyping followed by a cycle of interaction with the expert and modification of the consultation program. Frequently, the design must undergo major revision in order to accommodate the additional knowledge and strategy information that is revealed during this cycle.

Traditional software tools tackle problems at too low a conceptual level to be effective in debugging expert systems. AI systems provide their own control and data structures. The knowledge engineer wants to probe the system at times and places reflecting the nature of the high-level control regime in which the information has been structured; most

debugging facilities are oriented towards the control structure of the underlying implementation language [11].

In the past, knowledge engineers relied on textual messages which were printed on the terminal screen to monitor the internal interactions of the datatypes. Although these messages served their purpose, the amount of text produced could be overwhelming if the knowledge engineer requested a full trace. In the majority of cases, the user needed to scan the trace in order to find key pieces of information. In addition, some systems dynamically created datatypes as needed during the course of a consultation. A trace of these objects would reference them by internal non-mnemonic names that masked how they were related to other objects in the knowledge base.

In conventional systems it is all too easy to lose the debugging context when the editor is invoked. When a consultation is suspended and the knowledge engineer needs to edit various structures, the screen clears to enter the display editor. If a line editor is used, the information will scroll off the screen after a number of edit commands have been issued. It is helpful to reference debugging information during editing since there are a number of bugs which can crop up during a consultation session. Until recently, a knowledge engineer was forced either to commit the details to memory or to write them down. The window facility in graphical systems allows mode switching without losing the information about the current situation. Debugging can take place in one window with the editor being assigned to another window.

Graphics can enhance the amount and quality of information that can be extracted while monitoring an expert system. An inherent problem of large rule-based systems is the complex way in which the information structures interact, resulting in unforeseen side effects. Simulations of the datatypes during a consultation can increase the understanding of cause and effect relationships within the knowledge base. It is difficult to communicate these interdependencies fully with serial output. While debugging complex systems it is helpful to represent the execution of control tasks as well as reasoning chains and individual rules and parameters. These datatypes can be monitored in separate windows in order to partition the information so that it can be referenced quickly and easily.

In the future, debugging will also be complicated when some expert systems begin to take advantage of parallel processing architectures. Non-sequential execution makes it

imperative that a monitoring facility exist to help knowledge engineers isolate problems that occur only when running in parallel.

ONCOCIN's Dynamic Debugging Tools

ONCOCIN's graphical trace facility is organized so that the knowledge engineer can follow a consultation in terms of rules, parameters, control blocks, and contexts. Each datatype is monitored in a separate window located in a different section of the screen. Datatypes can be examined in one of two ways during a consultation. A datatype can be *traced*, which means that a message will be printed in specialized windows (see below) whenever it is executed or a value is concluded for it. Datatypes can also be *broken* which means that the system will suspend the consultation and allow the knowledge engineer to examine system status and variables before the desired datatype is evaluated. Datatype instances which are to be traced or broken are specified by the knowledge engineer using a series of menus. A second menu (with the keyboard as backup if there are too many choices) is used to enter the specific instances to be traced or broken. The consultation can be temporarily stopped either by specifying a datatype instance as a break point or by pressing a mouse button.[4]

After the trace and break lists have been specified, the knowledge engineer can start a consultation. Traced data structures are handled in the following way:

- Control blocks are displayed in a separate window and their execution is represented by inverting and then uninverting each step as it is carried out (see Fig. 5). If another control block is invoked, then a new window for that task is created and is offset from the tracing window for the first control block.
- A parameter tracing window shows the dynamic construction of the reasoning chain while it is chaining either forward or backward from a traced parameter (Fig. 6). This reasoning chain is represented as a graph with parameters and rules represented as graph nodes. The graph is updated constantly; each time a new rule is invoked, it is displayed as a new node in the graph. Failure of a rule is indicated by display of the label *failed* beside the rule in question. Success is indicated by continued growth of the reasoning chain.

---

[4]This feature is, of course, available only to the knowledge engineer and is disabled when the advice program is in regular use.

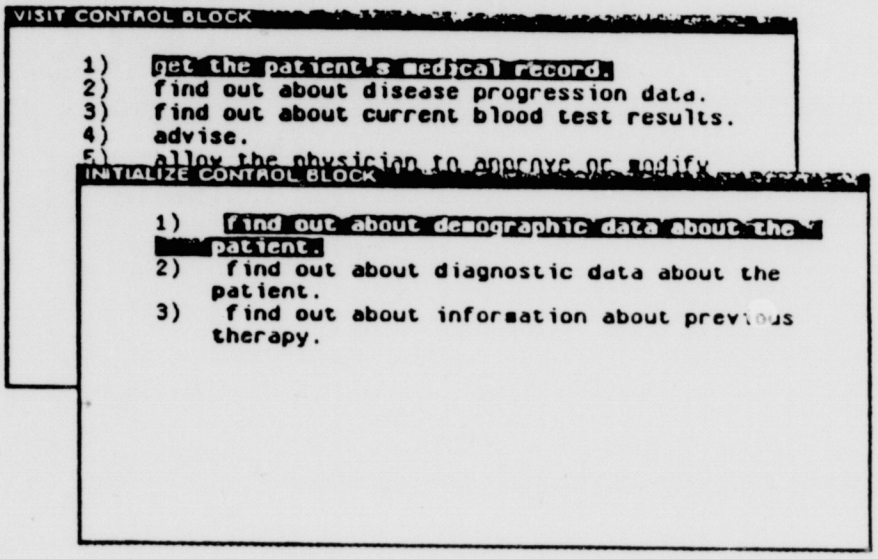

Figure 5: Control Blocks

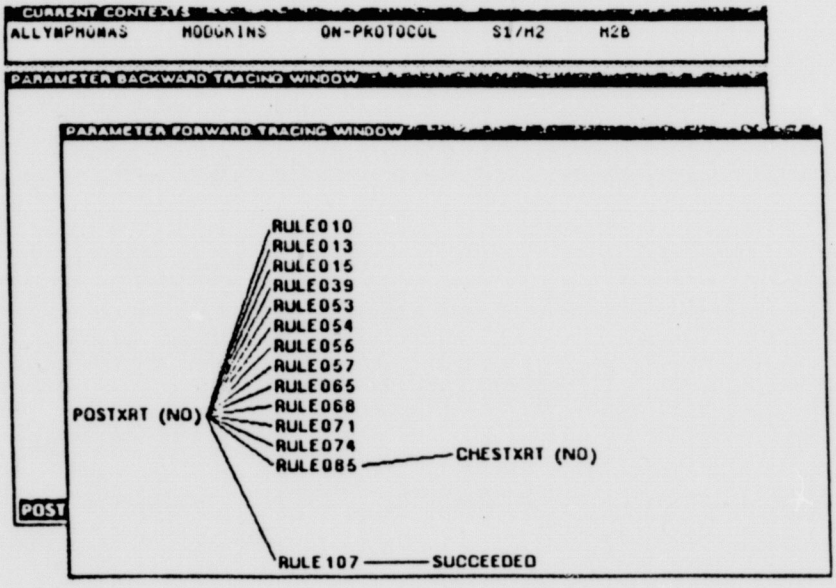

Figure 6: Parameter Tracing Windows and a Context Window

- Elsewhere on the screen, the rule currently being evaluated is displayed in a rule clause evaluation window (Fig. 7). Each clause in the rule's premise or action is inverted as it is executed. The result of the evaluation (failure or success) is

printed at the bottom of the window.

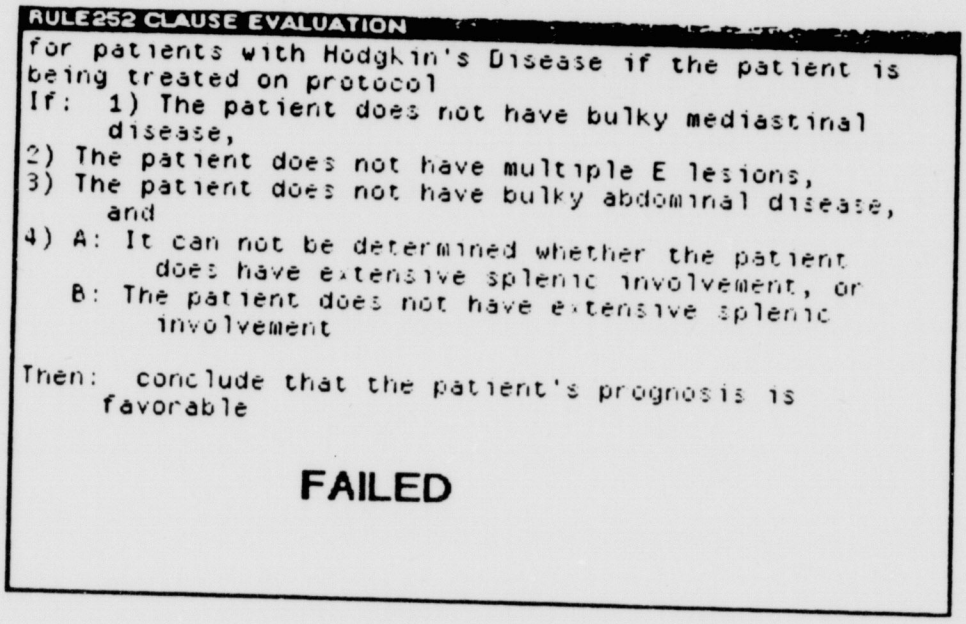

Figure 7: Rule Clause Evaluation Window

- During the course of a consultation, different contexts are assigned. Those contexts which remain constant during a consultation are displayed in one window (Fig. 6). Since the drug context takes on a number of different values during a consultation, however, these are hierarchically displayed in another window. As a new value is assigned, the corresponding node is inverted in the graph.

The knowledge engineer has a number of options available after breaking the consultation. These include asking how a parameter was concluded, asking how a rule was used, invoking a LISP break, editing a datatype instance, adding more datatypes to the break and trace lists, or continuing the consultation. If parameter justification is selected, a window appears explaining in English how the conclusion was made. In the majority of cases, a rule is cited as the source of the conclusion. The user can further examine the bold-faced text portions of the window. If rule justification is selected, a window appears which graphically displays how the rule was used.

Graphically monitoring ONCOCIN's data structures during a consultation detected some inefficiencies in the system not previously appreciated. One problem was repeatedly testing a boolean parameter whose value is already known. Another problem was the manner in which ONCOCIN's rules were tried. Some rules were meant to be used only forward

chaining, others only backward chaining. When ONCOCIN was trying to find the value for a parameter by backward chaining it also tried and rejected forward chaining rules. The use of a dynamic graphical debugging facility can therefore help identify sources of inefficiency (as well as frank errors) and thereby provide constructive suggestions for system revisions to improve overall performance.

Conclusion

In many respects, a graphical system is well suited to the task of information management. Much more information can be organized and displayed on the screen through the use of windows. The knowledge engineer has control over the amount of detail that is to be displayed and this control is *unobtrusive* and *intuitive*. Graphics techniques provide a greater breadth and flexibility for information display. The same information can be described with text or represented in a graph or diagram. The approach is especially well-suited for providing a comprehensive overview of a knowledge base, one that is hard to glean from written documentation.

How well these knowledge management tasks are accomplished will affect the development of expert systems, their performance on complex tasks, and, in turn, the technology's acceptance by its intended audience.

Acknowledgments

The authors would like to thank the following people for their assistance with this research or their comments on earlier drafts of the paper: Terry Barnes, Miriam Bischoff, Larry Fagan, Jay Ferguson, Bob Joyce, Christopher Lane, Curt Langlotz, and Jock Mackinlay.

References

1. Tsuji, S. and Shortliffe, E.H., Graphical Access to the Knowledge Base of a Medical Consultation System," *Proceedings of the AAMSI Congress '83*, American Association for Medical Systems and Informatics, May 1983, pp. 551-555.

2. Bischoff, M.B., Shortliffe, E.H., Scott, A.C., Carlson, R.W., and Jacobs, C.D., Integration of a Computer-Based Consultant into the Clinical Setting," *Proceedings of the 7th Symposium on Computer Applications in Medical Care*, Symposium on Computer Applications in Medical Care, October 1983, pp. 149-152.

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

4. Hollan, J.D., Hutchinson, E.L., and Weitzman, L., STEAMER: an interactive inspectable simulation-based training system," *AI Magazine*, Vol. 5, 1984, pp. 15-24.

5. Stefik, M., Bobrow, D.G., Mittal, S., and Conway, L., Knowledge Programming in Loops," *AI Magazine*, Vol. 3, No. 3, Fall 1983, pp. 3-13.

6. Shortliffe, E.H., Scott, A.C., Bischoff, M.B., Campbell, A.C., van Melle, W., and Jacobs, C.D., ONCOCIN: An Expert System for Oncology Protocol Management," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, International Joint Conference on Artificial Intelligence, August 1981, pp. 876-881.

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

8. Teitelman, W. and Masinter, L., The Interlisp programming environment," *IEEE Computer*, Vol. 14, 1981, pp. 25-34.

9. Suwa, M., Scott, A.C., and Shortliffe, E.H., An Approach to Verifying Completeness and Consistency in a Rule-Based Expert System," *AI Magazine*, Vol. 2, No. 3, Fall 1982, pp. 16-21.

10. Sheil, B., Power Tools for Programmers," *Datamation*, Vol. 29, No. 4, February 1983, pp. 131-144.

11. Model, Mitchell L., *Monitoring System Behavior In a Complex Computational Environment*, PhD dissertation, Stanford University, January 1979.