DEBUGGING EXPERT SYSTEMS USING A DYNAMICALLY CREATED HYPERTEXT NETWORK

Craig D. B. Boyle and John F. Schuette

Dept. of Computer Science
Texas A&M University
College Station, Texas 77843-3112
internet: craig@cs.tamu.edu

Abstract. The labor-intensive nature of expert system writing and debugging has motivated this study. Our hypothesis is that a hypertext based debugging tool is easier and faster than one traditional tool, the graphical execution trace.

HESDE (Hypertext Expert System Debugging Environment) uses Hypertext nodes and links to represent the objects and their relationships created during the execution of a rule based expert system. HESDE operates transparently on top of the CLIPS rule based system environment and is used during the knowledge-base debugging process. During the execution process HESDE builds an execution trace. Use of facts, rules, and their values are automatically stored in a Hypertext network for each execution cycle. After the execution process the knowledge engineer may access the Hypertext network and browse the network created. The network may be viewed in terms of rules, facts and values.

An experiment was conducted to compare HESDE with a graphical debugging environment. Subjects were given representative tasks. For speed and accuracy, in eight of the eleven tasks given to subjects HESDE was significantly better.

INTRODUCTION

Knowledge based systems are being increasingly used in diverse environments, writing such a system is no longer a research issue. However, the labor intensive and error-prone nature of their development has spurred researchers to examine development tools.

The hypothesis of this study is that a hypertext (Conklin, 1987; Nielsen, 1990) based debugging tool is more productive than one traditional tool, the graphical execution trace. In this context, the definition of a "productive" system is one that:

- is easy to learn and use,
- is fast to use,
- helps the knowledge engineer "understand" the expert system, and
- is relatively insensitive to changes in the particulars of the expert system's execution.

This definition of productivity is motivated by several factors. As with traditional programs, expert systems evolve over a potentially long lifetime. Over this lifetime, personnel not involved with the original development of the project will eventually be used to maintain the system. Their tools should make it relatively easy for the new knowledge engineer to grasp the processing of the system. Also,

apparently minor changes in an expert system may bring about drastic changes in the course of a consultation. Depending on the tool, such a change may make the debugging task harder for the knowledge engineer; it would be helpful if the debugging tool presents the knowledge engineer with a familiar landscape.

It is not hard to imagine a hypertext document that aids the knowledge engineer in his tasks. The objects of the expert system (facts, rules, etc.) could be represented by nodes. Links between the nodes could represent the causal relationships between rules and facts and the antecedent relationships between facts and rules. If such a network were built by an inference engine during a consultation, the knowledge engineer could browse the network afterwards to answer his questions about the session.

The research presented here investigates the potential role for hypertext as a tool to debug knowledge based expert systems. This discussion has several parts; first the expert system debugging task will be considered and differentiated from knowledge acquisition tools. This introduction to debugging and maintaining expert systems will provide some important insights into the characteristics of appropriate tools. For instance, what kinds of information does the knowledge engineer need when debugging his expert system? The traditional tool for monitoring an expert system's execution is the graphical execution trace. This trace is usually presented as a graph indicating the various relationships between specific facts and rules used in the consultation.

Previous research points to a need for effective expert system maintenance tools and hypertext is a candidate medium for those tools. The third section of this paper specifies the design of a novel hypertext network, named HESDE (Hypertext Expert System Debugging Environment), for debugging expert systems. This hypertext network is built automatically by the expert system's inference engine which is a modified version of the CLIPS engine (CLIPS, 1989). The network represents the important objects in the consultation via nodes and their interrelationships via links between the nodes. After the consultation halts, the knowledge engineer simply browses through the network to answer his debugging questions. The network provides links among facts, rules, agenda information and cycle number.

A critical question is whether this hypertext network is more useful to knowledge engineers than tools already in use. The fourth section describes an experiment conducted to evaluate that system's effectiveness as a debugging tool.

BACKGROUND

Significant expert systems continually evolve (Bachant et al, 1984; Smith 1984). Therefore, there should be a methodology and a set of tools for the maintenance and debugging of expert systems. Because of their non-procedural nature, however, expert systems debugging is not always amenable to the "snapshot" strategies utilized by debugging methods and tools commonly used for conventional programming languages. Different tools must be designed for the maintenance of expert systems.

The important questions to designers of these tools should be: What kinds of things do these tools have to do? What kinds of information do they have to convey? Traditional debugging tools tell us about the objects in a running program: instructions and data. Analogous tools for expert systems should do the same; they should tell us something about the objects in an expert system consultation: the agendas, rules, and facts. Neches et al have designed a system

that uses execution traces, domain knowledge, and knowledge about expert systems themselves to provide more useful explanations (Neches et al, 1985). They conclude that this "approach also offers other benefits related to development and maintenance." These benefits mostly are a result of the separation of the different types of knowledge (domain, expert system, etc.), but some benefit is also provided by the improved explanations. Because of this relationship between explanation and maintenance, it is useful to summarize the results of the research into different types of explanations.

There are various methods available to allow an expert system to explain its reasoning. These include simply having "canned text" in the system to handle expected questions, using a execution trace to deduce the pattern of reasoning, and building models of the problem domain to allow higher level explanations (Swartout, 1981; Swartout, 1983). Regardless of the method used, the designer must have an idea of what kinds of questions the user (or knowledge engineer) might ask.

Question Types

Swartout (Swartout, 1981) describes three types of questions that could be asked of an explanation system. The first can be called descriptive questions; they are asked to clarify the methods used by the system. The second type of question is the justification question: "Why did you conclude fact x?" In a debugging context, this question may take the form of "What rule set the value of fact x to y?" Except in trivial cases, this type of question is difficult to answer. Finally, Swartout's third type of question could be called a clarification question: "What do you mean by term x?". To answer these questions requires a significant amount of domain knowledge.

Answer Types

Gilbert (Gilbert, 1987) divides his answer types into three groups according to the source of the answer. These groups are the theory group, the domain group, and the case level group. The theory and domain groups correspond roughly to Swartout's clarification question type. Answers from the case level group deal with issues related to particular consultations and are therefore the most interesting to a debugging knowledge engineer. The case level answer types and their relevance to the debugging process include the following answer types:

An *instantiation answer* tells whether a certain fact exists in the knowledge base. Such information is commonly needed by a knowledge engineer to solve debugging problems (e.g., "I expected fact x to get asserted during the consultation. Did it?")

A *classification answer* is a decision made about some situation. Such an answer is usually the end product of the consultation and is obviously of interest to the knowledge engineer.

A *prescription answer* may be the by-product of a diagnosis expert system. They are typically generated by additional rules after the diagnosis (classification) is made. Again, these answers are of interest to the debugging knowledge engineer because they help indicate the visible correctness of the system.

A *justification answer* tells how some conclusion was reached. A closely related answer type is the *antecedent answer*; it describes events that lead to the conclusion. The answer to both question types can be found by examining a trace of the rules fired and facts asserted in the consultation. This is of great usefulness to the knowledge engineer because he can use these answers to verify that facts are asserted for the correct reasons, or can perhaps determine why some expected event did not occur.

HYPERTEXT

Hypertext dates back to Vannevar Bush's memex (Bush, 1945). Recent advances in personal workstation technology (particularly high-performance displays and windowing systems) have enabled hypertext to become a practical solution to many information management problems.

The Hypertext Expert System Debugging Environment – an Overview

The hypertext network introduced here is a new type of debugging tool; it is a Hypertext Expert System Debugging Environment (HESDE). In general, a HESDE is constructed automatically during the execution of some expert system's inference engine; the particular HESDE described in this paper is built by an forward chaining inference engine based on the engine in CLIPS. HESDE transparently records the execution of CLIPS, noting rule firings, fact instantiations and their interrelationships. After the execution of CLIPS, HESDE presents a browsable version of the execution environment to the user. Figure 1 shows the HESDE environment. Once CLIPS execution is complete the user may browse the network. The next sections will explain the functionality, structure and use of the network.

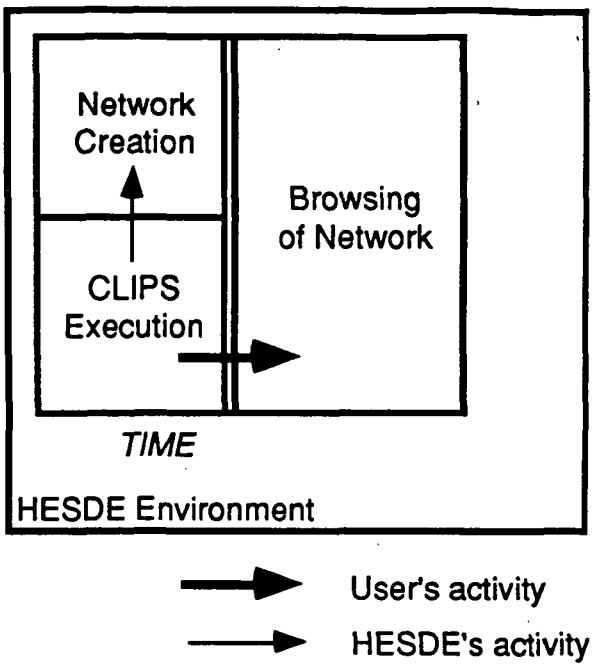

Figure 1. HESDE Environment

The knowledge engineer browses the HESDE network after execution of the expert system to find information about the objects and relationships created during that execution. HESDE is not just a snapshot of the system state at the end of execution, but a complete, cycle-by-cycle and inference-by-inference record of activity. Ideally, a HESDE would operate concurrently with the activity of the expert system, the present system only allows a post-execution analysis.

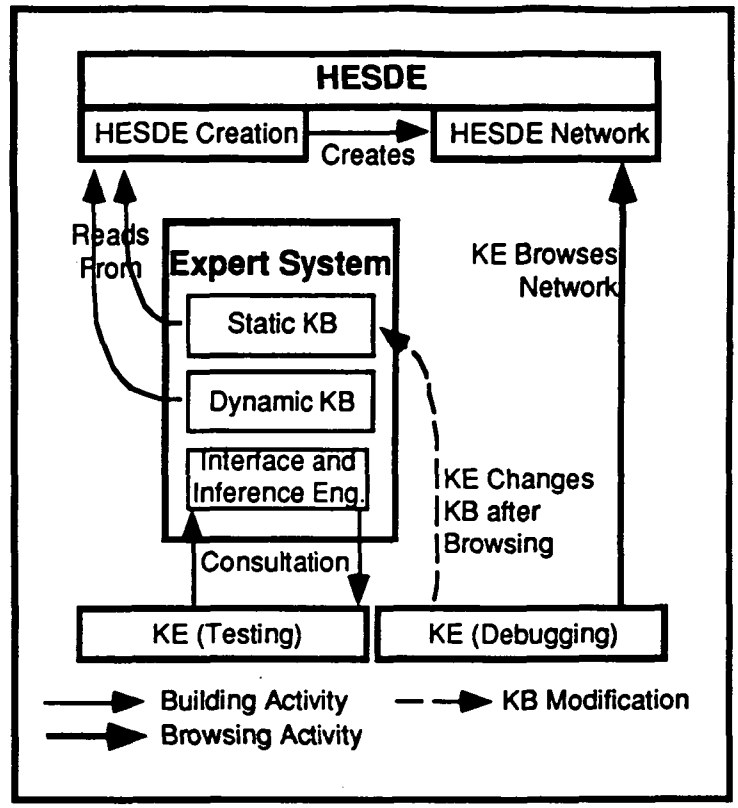

Figure 2. Overview of HESDE's activity

Hypertext model used by HESDE

A node in the HESDE system uses the card model used in hypertext systems such as NoteCards[1] (Xerox, 1985) and HyperCard[2] (Apple, 1990). In the HESDE systems, the node appears as a fixed-size window arbitrarily located on the screen. The card is smaller than the display, so multiple probably overlapping cards are visible simultaneously. The ability to display multiple cards simultaneously is important; in the HESDE system, it allows the knowledge engineer to leave one card on the screen while going to find another. If only one card were visible at a time, the knowledge engineer would be forced to remember the contents of the earlier card reducing the utility of the tool. Figure 3 shows a typical card.

[1]NoteCards is a trademark of Xerox Corporation.
[2]HyperCard is a trademark of Apple Computer, Inc.

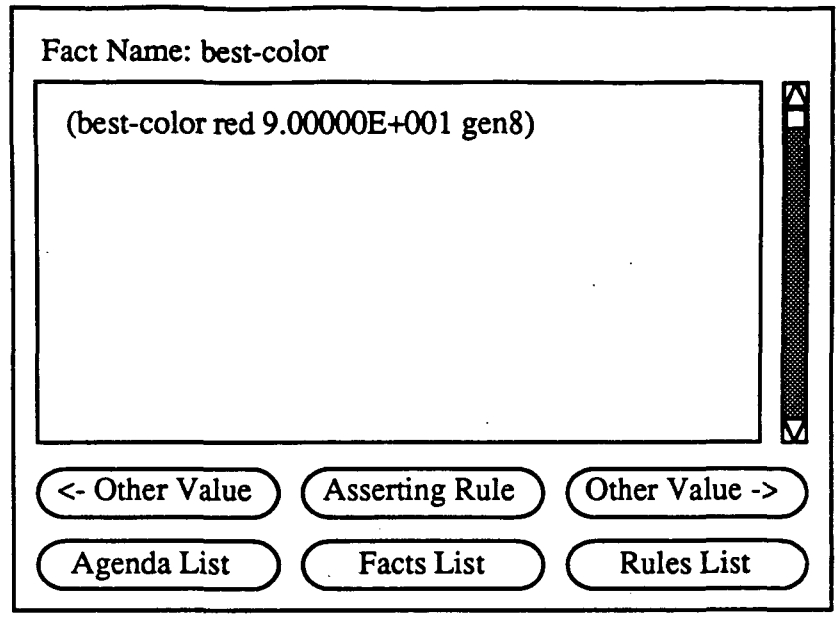

Figure 3. A typical card in HESDE[3]

Link Issues

All links are represented explicitly as buttons on the card. These buttons may appear "directly on" the card or as a part of the scrollable field. The HESDE system consist of a Hypertext browser and a Hypertext network creator. The network created by the HESDE is static and immutable. It has no capability for user-authoring–i.e. it is not (currently) possible for the user to change rules permanently or see the effect of fact changes on network status.

HESDE Network

We will now explain the structure and use of the HESDE network. Firstly an overview of the network structure and contents is presented. Secondly the three subnets and their interconnections are described in detail. Finally, the performance of the HESDE is briefly compared to that of a graphical execution trace.

Network Overview

The network created by HESDE is a directed graph containing several node types. The different node types are summarized in Table 1.

[3]The figures presented in this paper are schematic diagrams for clarity's sake.

Node type	Displays	Link targets
RIN	rule instance	master rule
		previous rule instance
		subsequent rule instance
		rule agenda
		antecedent facts
RMN	rule prototype	rule instance
RLN	list of rules	master rules
FIN	fact instance	previous fact instance
		subsequent fact instance
		asserting rule instance
FLN	list of facts	fact instances
AIN	rules on agenda	rule instances
ALN	list of agendas	agenda instances

Table 1. Node types and functions

The node types defined for the HESDE correspond to the basic objects in an expert system: the rules, facts, and agendas. Because this HESDE is based on CLIPS and because these are the only types of objects in CLIPS, the above list of node types is sufficient. New node types and relationships may have to be added to represent features supported by other expert system tools.

Subnets

Although the HESDE network is a single connected directed graph, it is convenient to describe it in terms of three subnets: the rules subnet, the facts subnet, and the agenda subnet. This division is made because at the highest level, the system presents the consultation as lists of the major conceptual objects in the consultation: rules, facts, and agendas.

These subnets each have a single node that serves as a "root" node for that subnet. In each subnet, this root node is essentially one of the lists mentioned above (rules, facts, or agendas). All nodes in the HESDE net contain links to each of these root or list nodes. This cross-linkage provides an efficient way for a browsing knowledge engineer to access a standard reference point in any particular subnet; getting lost is a common problem in large hypertext networks (Conklin, 1987).

The rules subnet

The first subnet in the HESDE net is the rules subnet. It consists of three types of nodes: Rule Master Nodes (RMN), Rule Instance Nodes (RIN), and a Rule List Node (RLN).

As the knowledge base is loaded, the system creates a RMN for each rule. At the outset, this node contains only the text of the rule in question. See Figure 4 for an example of an RMN.

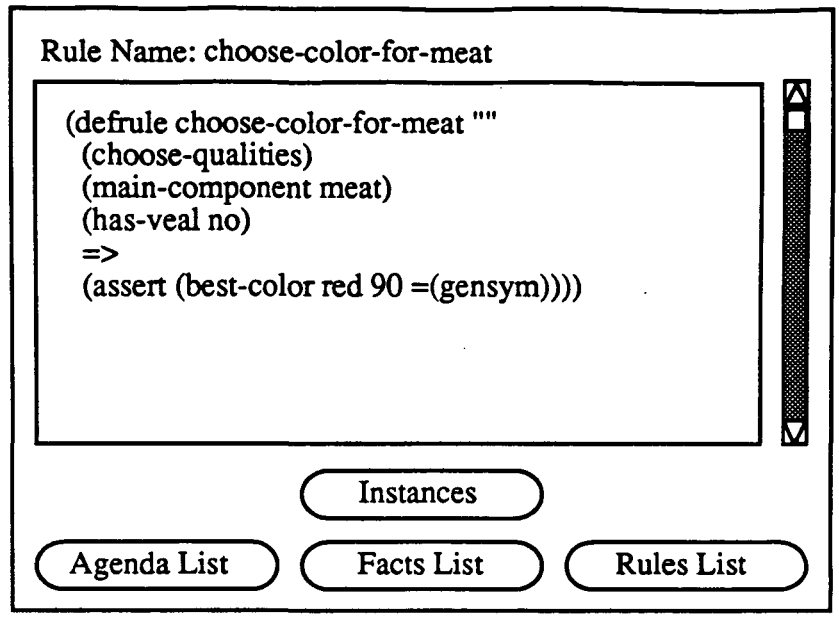

Figure 4. A typical RMN

Once the consultation is started, an RIN is created for every rule instance enrolled in an agenda. An RIN is similar in content to an RMN, but has some additional content. In particular, links to fact nodes are imbedded in appropriate places of the rule text. (The creation of HESDE nodes corresponding to facts in the knowledge base will be explained later.) These links are provided to allow the knowledge engineer to quickly determine the value of a fact bound to a particular rule. RIN's and RMN's are interconnected in the following manner; the first instantiation of a particular rule creates a new link on that rule's RMN that points to the first RIN for that rule. As new instances of that rule is created, the new RIN's are prepended to the chain of RIN's already connected to that RMN. This provides a path from the master rule to the instance rules. The reverse path is provided by a link in every RIN that points back to the RMN for the appropriate rule. This interconnection strategy is illustrated in Figure 5.

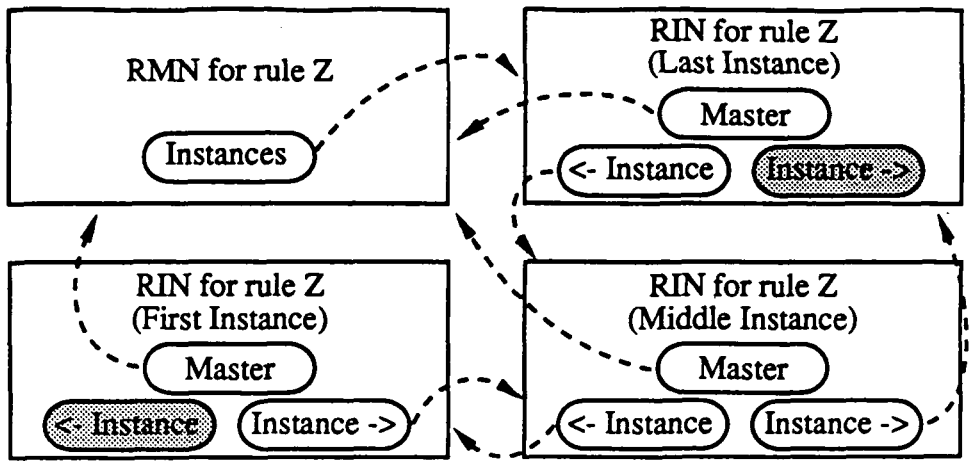

Figure 5. Some rule node interconnections

There are two reasons for providing both master and instance rule nodes. First, we need the RMN's because we can't rely solely on RIN's; it is common for a knowledge engineer to need information about a rule that was never instantiated. For instance, he may ask "why didn't rule x fire?" Without an RMN, there would be no evidence whatsoever of an uninstantiated rule. Second, the RIN's are necessary because the system needs to be able to keep track of the history of the consultation; any given rule may have multiple instantiations during the consultation (including simultaneous multiple instances), so a single node for each rule will not suffice.

A secondary use for RMN's in future systems may be to provide a logical point at which the knowledge engineer may edit the definition of a rule. The current HESDE system provides no editing capability, but such an extension would be a logical one.

The RLN is the last node type in the rules subnet. It serves as the root of the rules subnet and contains a link for every RMN created by the knowledge base loading process. These links are alphabetized according to rule name for ease in searching. As mentioned earlier, every node in the HESDE network has a link to the RLN to allow quick access to this reference point. Figure 6 shows a RLN and its links to other rule nodes in the network.

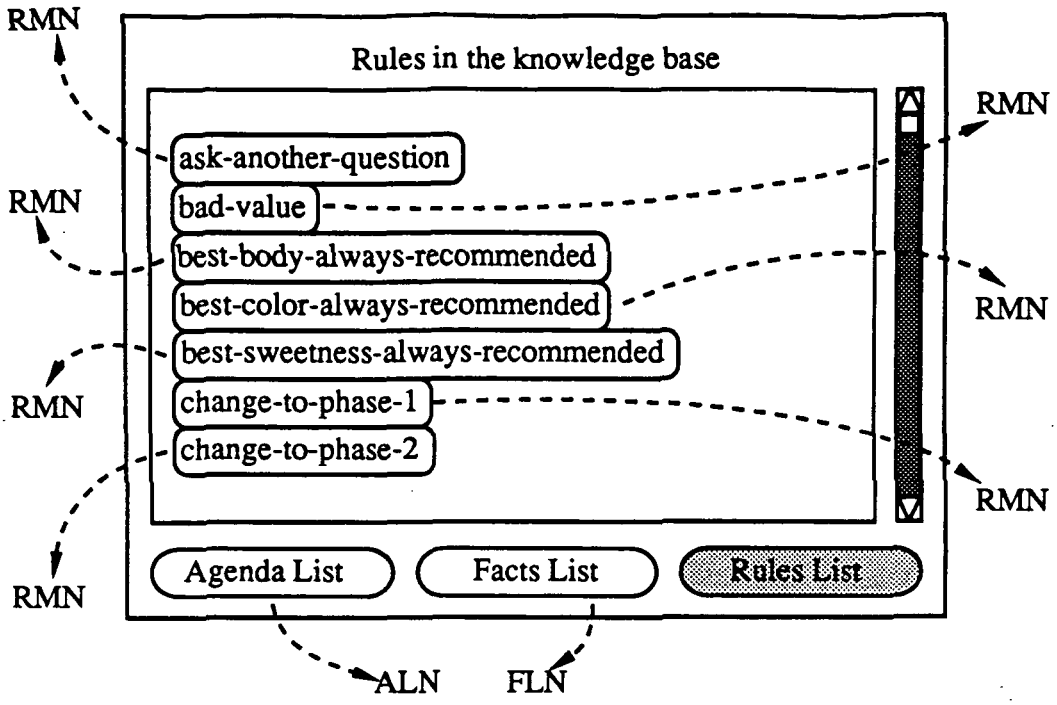

Figure 6. A RLN and its place in the network

The facts subnet

The second subnet is the facts subnet. There are just two types of nodes in the facts subnet: Fact Instance Nodes (FIN) and a Fact List Node (FLN).

A new FIN is created for every fact asserted during a consultation. Each FIN contains a printed representation of the fact instance in question. Before explaining the facts subnet further, it is important to define the HESDE's concept of a fact.

The interpretation attached to facts and their representations is clearly system dependent. For instance, facts in the KEE[4] system typically represent the characteristics of some object in a class hierarchy (Intellicorp, 1987). On the other hand, in CLIPS, facts are essentially just arbitrary lists of symbols (CLIPS, 1989). As will be detailed in the implementation section, the HESDE described here is based on the CLIPS expert system building tool. Therefore, the appropriate model of a fact is a list of symbols.

There is no actual semantic network connecting facts in a CLIPS knowledge base. Instead, the relations between facts are implicit and are enforced by conventional CLIPS programming style. In a CLIPS knowledge base, the first element of a fact's list representation usually names the fact. Any subsequent elements in the list are viewed as slots or values for the entity named in the first element. For example, the CLIPS facts (temperature 350) and (temperature 400)

[4]KEE is a registered trademark of Intellicorp, Inc.

may suggest two readings of a thermometer or two recommendations for a roasting temperature.

Therefore, using this fact model, the printed representation of a fact in a FIN is simply a display of that fact's list of elements. Figure 7 illustrates a typical FIN.

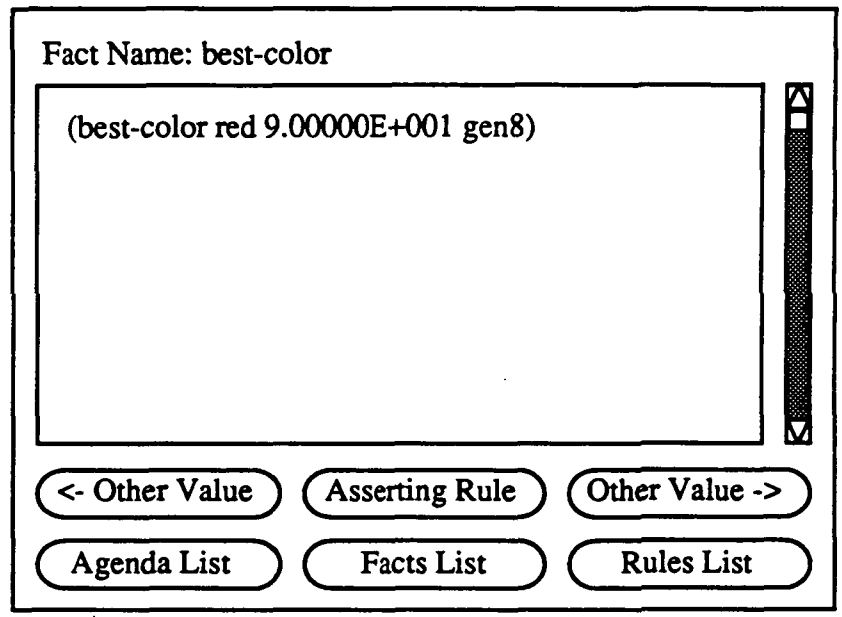

Figure 7. A FIN

Note that, in general, there may be more than one instance of the same fact in the knowledge base (i.e., more than one fact with the same name or first element). Whenever a new FIN is created, a check is made to see if there are any previous versions of the same fact in the knowledge base. If there are, two links are added to the facts subnet. First, a link is made from the new FIN to the pre-existing FIN. This link may be followed by the knowledge engineer to discover the "history" of a given fact. Second, a link is made from the older FIN to the new one; this link simply provides a return path to more recent versions of a fact.

As in the rules subnet, there is a root or list node used to provide direct access to any FIN in the facts subnet. This is the role of the FLN. After the first version of a fact is asserted and its FIN is created, a new link is added to the FLN. The target of this link is simply the newly created FIN. The buttons representing these links from the FLN are displayed in alphabetical order to allow the knowledge engineer to quickly find the FIN for a specific fact. Figure 8 shows an example of a FLN.

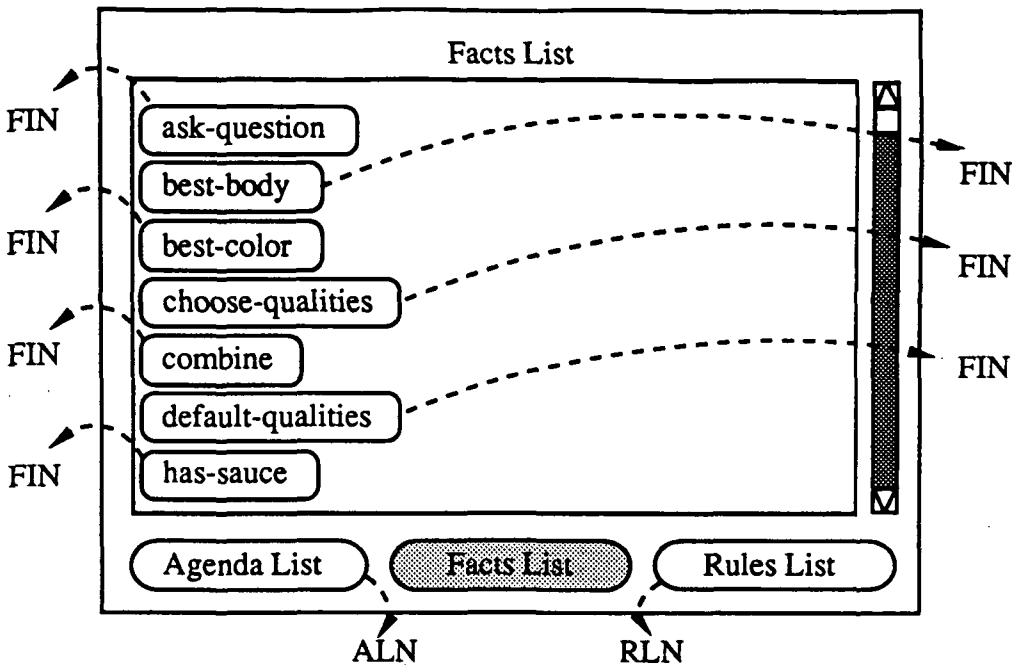

Figure 8. An example of a FLN

As discussed above, there may be more than one instance of a given fact at one time. The FLN lists only one link per fact name regardless of how many versions of that fact there may be. This restriction prevents the number of links from the FLN from growing too quickly. A fact link from the FLN always points to the most recently asserted version of that fact. If the knowledge engineer needs to examine a previous version, it is always available through the "previous version" links of the more recent versions of that fact. The method of interconnecting nodes in the fact subnet is illustrated in Figure 9.

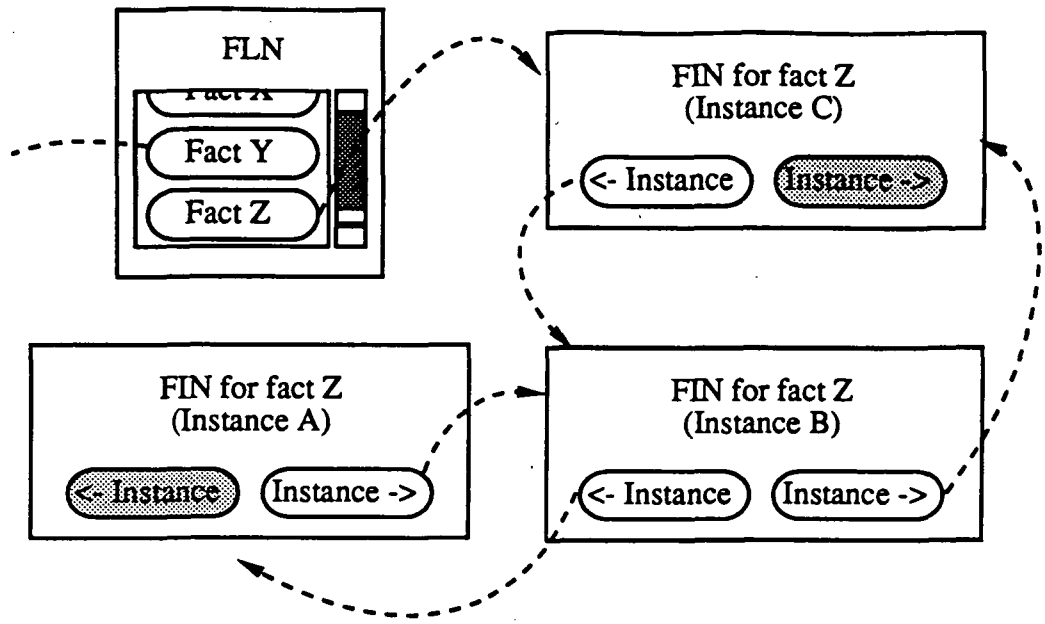

Figure 9. Fact interconnections

The agenda subnet

In an expert system, the agenda is the list of rule instances that are eligible to fire during a given cycle. Agendas are sometimes called conflict sets (Brownston et al, 1985). The agenda subnet is responsible for maintaining copies of all the agendas generated by the consultation. This information is valuable to the knowledge engineer because it helps provide the answer to questions such as "what rules were under consideration at point x and which one was fired?"

In the HESDE network, the agenda subnet consists of two types of nodes: Agenda Instance Nodes (AIN), and the Agenda List Node (ALN). In each recognize-act cycle of the inference engine, the system generates a new AIN for the current agenda. That AIN contains a link to every RIN that corresponds to a rule instance in the current agenda. Figure 10 illustrates such a node.

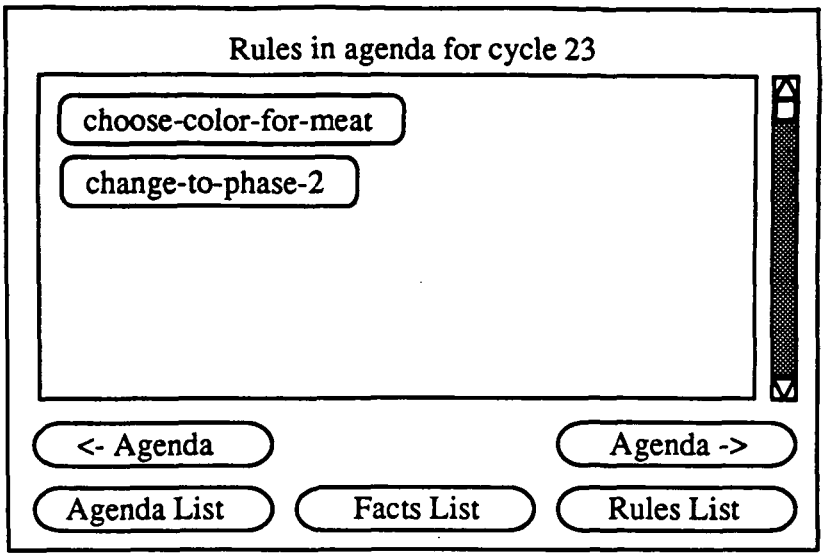

Figure 10. An AIN

So that all AIN's are accessible to the knowledge engineer, an ALN is created at the beginning of the consultation. As each new agenda is created by the inference engine, a link to its AIN is added to the links on the ALN. This ALN is presented as one of the roots of the hypertext network.

Subnet interconnections

Most links have their source and target in the same subnet. The most useful links in the network, however, are those that have their source and target in different subnets. This is because these links show the relationships between different types of objects: rules and facts, for instance. Now that all three subnets have been explained, these critical links can be described in detail.

The first class of subnet interconnection links are the agenda-rule links. As described above, they link AIN's to the RIN's corresponding to the rules in that agenda.

The next class of interconnection links are the rule-agenda links. In the description of the agenda subnet, the links from an AIN to a RIN were described. To provide the return link, in every RIN, there is a link from that RIN to the AIN corresponding to the agenda of which that particular rule instance is a part. This link allows the knowledge engineer to quickly determine of which agenda a particular rule instance is a member. Through the AIN, he can also determine what other rules were on the agenda at the same time as a particular rule instance. Figure 11 shows examples of agenda-rule and rule-agenda links.

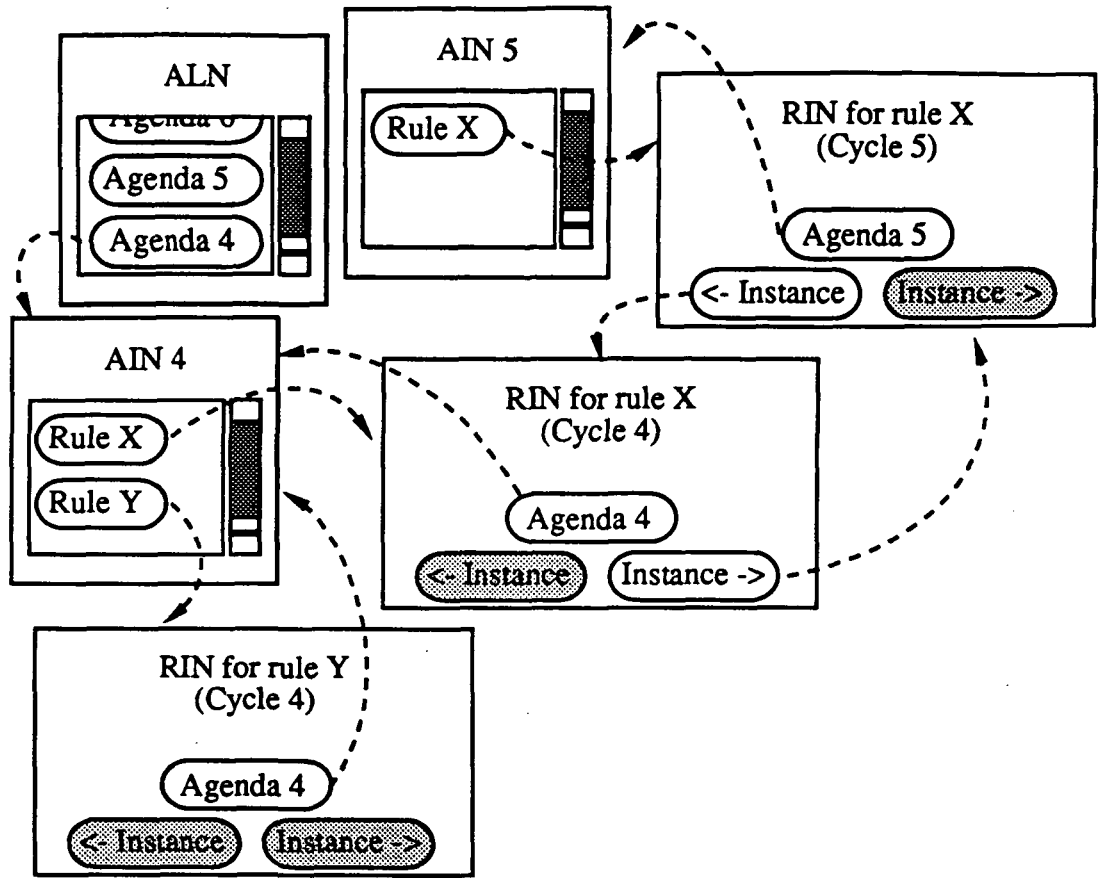

Figure 11. Agenda and rule interconnections

Another class of interconnection links is the fact-rule links. As each FIN is created, a link is established between that fact's node and the RIN of the rule that asserted that fact. By following such a link between a fact node and a rule node, a knowledge engineer can determine the rule responsible for a particular fact.

Finally, rule-fact links are the last class of interconnection links. These links allow the knowledge engineer to examine why a particular rule instance was eligible to fire. A rule instance is simply a binding of a rule with some set of facts that match the conditions named on the left hand side of that rule. As discussed above, each RIN corresponds to a specific rule instance and contains the source text of a particular rule. When a RIN is created, every mention of a fact in the left hand side of the rule is turned into a link to the FIN for that fact. By following these rule-fact links, the knowledge engineer can easily determine the value of that fact, what rule asserted it (through that FIN's fact-rule link), etc.

EVALUATION

To show the usefulness of HESDE, it was compared with a popular conventional tool. For comparison, we chose KEE (Intellicorp, 1987) which uses a graphical overview as the principal means of viewing a network. For

experimentation, six users were assigned to KEE and HESDE respectively. They were asked to simulate the debugging of a pre-defined expert system by performing a set of typical tasks.

The task list was compiled based on the kinds of information a debugging knowledge engineer might need. For instance, tasks 1, 4, 8, and 11 correspond to Gilbert's (Gilbert, 1987) justification answers. Tasks 6 and 10 are related to his antecedence answer type. The other tasks deal with the basic programming concepts: tasks 2 and 9 relate to control of rule firing; tasks 3, 5, and 7 just correspond to the knowledge engineer's examining the text of a rule.

Results

For eight of the eleven tasks HESDE proved to be statistically significantly better in terms of speed and accuracy. There was no significant difference in incidental learning between the subject groups.

Task completion time was measured as the time between clicks of the "go" and "stop" buttons with deductions made for the subject's looking back at the script. The deductions for re-reading the script were determined with a stopwatch during the experimental trial. In the case of the KEE subjects, additional deductions were made for machine response time. These deductions were determined using the video record of each subject's session. Figure 12 presents the mean task completion times by task.

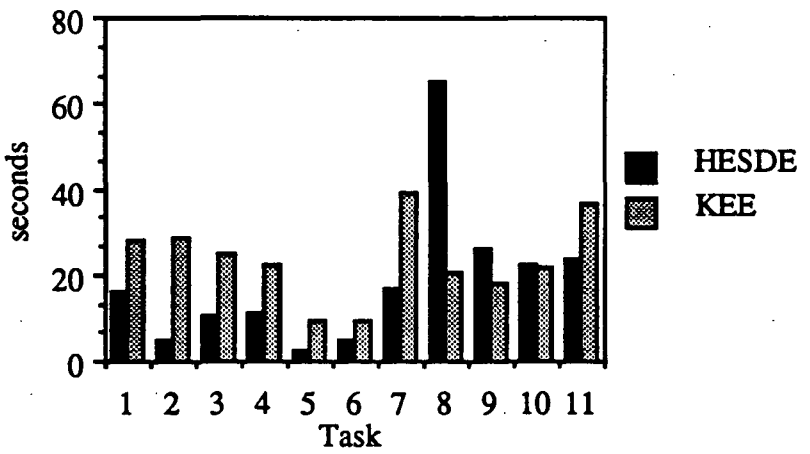

Figure 12. Graph of mean task completion times

These results indicate that the HESDE system is significantly faster ($p < 0.05$) for tasks 1-7 and marginally faster ($p < 0.10$) for task 11. KEE is significantly faster for task 8 and marginally faster for task 9. Task 10 seems to take the same time in both systems.

CONCLUSION

HESDE provides a robust and flexible hypertext-based environment for viewing and debugging an expert system's rule base. Testing has shown that HESDE outperforms at least one commercial expert system's debugging environment. We believe that HESDE will best be used as a complement to other debugging tools rather than a replacement.

What generalizations can be made from these results? For instance, to what extent are the observed differences due to weaknesses in the particular graphical trace studied here? This is unlikely; the problems with the KEE graphical trace are inherent to all graphical traces. These are the need to search through the trace space instead of directly accessing elements, the inability for the user to keep a previously visited location on the screen, and the sensitivity of the trace to changes in the consultation due to its spatial representation.

REFERENCES

Apple Computer, Inc. (1990) *HyperCard User's Guide*. Apple Computer, Inc., Cupertino, CA, 1990.

Bachant, J., and McDermott, J. (1984) R1 revisited: Four years in the trenches. *AI Magazine 5*, 3 (Fall 1984), 21-32.

Bush, V. (1945) As we may think. *Atlantic Monthly*, July 1945, 101-108.

Brownston, L., Farrell, R., Kant, E., and Martin, N. (1985) *Programming Expert Systems in OPS5*. Addison-Wesley, Reading, Mass., 1985.

CLIPS (1989) Artificial Intelligence Section, Johnson Space Center. *CLIPS Reference Manual*. NASA.

Conklin, J. (1987) Hypertext: An introduction and survey. *IEEE Computer 20*, 9 (September 1987), 17-41.

Gilbert, G. (1987) Question and answer types. Research and Development in Expert Systems IV; Moralee, D., ed., 1987.

IntelliCorp, Inc. (1987) *IntelliCorp KEE Software Development System Rulesystem3 Reference Manual*.. Intellicorp, Inc., 1987.

Neches, R., Swartout, W., and Moore, J. (1985) Explainable (and maintainable) expert systems. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Volume 1, 1985, 382-398.

Nielsen, J. (1990) *Hypertext and Hypermedia*, Academic Press, 1990.

Smith, R. (1984) On the development of commercial expert systems. *AI Magazine 5*, 3 (Fall 1984), 61-73.

Swartout, W. (1981) Explaining and justifying expert consulting programs. In *Proceedings of the Seventh Annual Joint Conference on Artificial Intelligence*, August 1981, 815-822.

Swartout, W. (1983) XPLAIN: A system for creating and explaining expert consulting systems. *Artificial Intelligence 21*, 3 (September 1983), 285-325.

Xerox Special Information Systems. (1985) *NoteCards Release 1.2k Reference Manual*.. Xerox Corporation, Pasadena, CA, 1985.