Volunteering Information--
Enhancing the Communication Capabilities of Knowledge-Based Systems

Gerhard Fischer
Curt Stevens

CU-CS-367-87 June 1987

Department of Computer Science
Campus Box 430
University of Colorado
Boulder, Colorado 80309

This research supported by: Grant No.DCR-8420944 from the National Science Foundation, Grant No.MDA903-86-C0143 from the Army Research Institute and a grant from Microelectronics and Computer Technology Corporation (MCC),Austin.

Volunteering Information -- Enhancing the Communication Capabilities of Knowledge-Based Systems

Gerhard Fischer and Curt Stevens

Department of Computer Science and Institute of Cognitive Science
University of Colorado, Campus Box 430
Boulder, CO 80309

Abstract

Cooperative problem solving systems support the solution of tasks which cannot be solved by the human or the computer alone. These systems need to be knowledge-based and require flexible communication paradigms allowing natural communication with both experts and novice users of the system. Natural communication (quite different from natural language) has to support mixed-initiative dialogues where information can be volunteered by the system and the user.

In this paper, we present prototypical systems which assist users in rebooting a computer. REBOOTER is a rule-based system which guides the user with a strongly system-directed dialogue through this task. The use of this system has shown that the communication paradigm was too narrow to make it a worthwhile tool (especially for the expert user). The SYSTEMS ASSISTANT tries to overcome the noted shortcomings by allowing the users to interact with the system in a mixed-initiative dialogue, to volunteer information and to deviate from the system generated discourse structure.

1. Introduction

Our goal is to establish, both by theoretical work and by building prototypical systems, the scientific foundations for the construction of intelligent systems which serve as amplifiers of human capabilities and skills. A prerequisite for intelligent systems is that we understand the information processing possibilities and limitations of the human *and* the computer. Our systems should not only be significant as technical achievements in computer science, but also because they are based upon principled analyses of how one can best help people to cope with complex information systems.

Knowledge-Based Systems (KBS) and Human-Computer Communication (HCC) are two crucial research areas for these goals. We are especially interested in understanding the possibilities of pursuing these two research areas together. The rationale for this approach is that on the one hand effective human-computer communication is more than creating attractive displays on a CRT screen: it requires providing the computer with a considerable body of knowledge about the world, about users and about communication processes [Fischer 83]. On the other hand the use of knowledge-based systems will be severely limited if we are unable to eliminate the *communication bottleneck*.

After characterizing general communication paradigms, this paper examines one aspect of this approach, the design of knowledge-based systems and their communication capabilities to allow the volunteering of information by the user of the system. Being able to volunteer information, users of a knowledge-based system are no longer at the mercy of an unseen reasoning component that dictates the order in which information is absorbed by the system. When combined with a data driven rule base, users are offered an opportunity to *actively* use a system and direct it according to their goals.

2. Communication Paradigms in Knowledge-Based Systems

The use of knowledge-based systems will be severely limited if we are unable to eliminate the communication bottleneck. The main reason that knowledge-based systems have not moved beyond the research state has primarily been their limited communication capabilities (an example being the MYCIN system [Buchanan, Shortliffe 84]). The analysis of the DIPMETER system [Smith 84] has revealed that the user interface portion is the largest part (42 percent) of a knowledge-based system.

In this section, a framework for different communication capabilities is illustrated by defining "natural communication" and "mixed-initiative dialogues", characterizing different system architectures depending on the distribution of the *speaker/listener* role and discussing architectures, requirements and examples for systems which allow the system and/or the user to *volunteer advice.*

Natural Communication. *Natural Communication* is more than the ability to communicate in natural language. It is the ability to engage in a dialogue and when humans (e.g., a novice and an expert) communicate much more goes on than just the request for factual information. Novices may not be able to articulate their questions without the help of the expert, the advice given by the expert may not be understood and/or the advisee may request an explanation of it; each communication partner may hypothesize that the other partner misunderstood him/her or they may provide information which they were not explicitly asked for.

Natural Communication needs the right kind of user interface to support it, but it cannot be restricted to just the user interface. The underlying knowledge base must contain the needed knowledge and it must be structured in the right way.

Despite the fact that communication capabilities such as *mixed-initiative dialogues* [Carbonell1970a] have been found to be crucial for intelligent systems, the progress to achieve them has been rather modest. Limited natural language interfaces have often overshadowed the real shortcomings. The MYCIN system and the REBOOTER (see section 3) serve as good examples: they are based on the *consultation model* . From an engineering point of view, this model has the advantage of being clear and simple: the program controls the dialogue (much as a human consultant does) by asking for specific items of data about the problem at hand. The disadvantages are that it prevents the user from volunteering relevant data and it sets up the program as an "expert", leaving the user in the undesirable position of asking a machine for help.

The Speaker versus the Listener Role. Based on the *asymmetry* between human and computer, the design of the communication between humans and computers is a problem not only of simulating human-to-human communication but of engineering alternatives in the domain of interaction-related properties [Bolt 84]. Natural language should not be used for every application; in many cases it is not the preferred mode of communication [Bates, Bobrow 84].

Communication can be described in terms of the speaker and the listener roles. The speaker presents information (e.g., in the form of a question or as a request for action) which the listener tries to understand. It is often difficult to determine which role suits which agent best. We have argued that the listener role is always the more difficult one [Fischer 86], because the listener has to understand the problem based on the speaker's description.

Natural language interfaces are desirable, because the human is the speaker and can talk in her/his terms about a problem. Unfortunately this kind of natural language interface does not exist. The user is either forced to answer questions in simple terms or to learn to adapt to the *limited* natural language understanding capabilities of the system. In form-based systems, the system has the role of the speaker and it shows its understanding of the world to the user. Our work has been primarily guided by the belief that the user is more intelligent and can be directed into a particular context; this is why most of our interfaces are form-based.

Computer Systems volunteering advice. Humans often learn by receiving answers to questions which they have never posed. For example, if they see a sign that says *"Snow Tires Or Chains Required Beyond This Point"*, they have learned many things. They know that there is probably snow ahead on the road and that they can buy snow tires to eliminate the need for chains. This information is volunteered -- there is no need to ask for it.

To ask a question, one must know how to ask it, and one cannot ask questions about knowledge whose existence is unknown. We have developed programs (e.g., the active help system ACTIVIST [Fischer, Lemke, Schwab 85] and the LISP-CRITIC [Fischer 87]), which volunteer information and support the acquisition of information by chance. ACTIVIST looks a user (working with an editor) "over the shoulder", infers from user actions the plan which the user wants to achieve and compares it with its own plan. Information about the conjectured knowledge is stored in the model of the user. A separate tutoring module decides when to offer help and advice. The LISP-CRITIC enhances incremental learning of LISP and supports learning strategies such as learning on demand. It has knowledge about how to improve LISP programs locally, following a style as defined by its rules. The advice given is based on the hypothesized knowledge of the user contained in the system's model of the user. Additional tools are available to explain and illustrate the advice.

A number of things have been learned constructing these systems. Volunteered advice is most welcome if it is directly relevant to the problem or the task the user is working on. The major problem in systems of this kind is not to make them speak up but to keep them quiet most of the time. To achieve this requires elaborate knowledge structures (e.g., models of the users and tutorial strategies). In addition, users must have the control to ignore the volunteered information (they may already know it or they may regard it as not relevant) or turn the systems off altogether.

Constructing systems which volunteer information creates a number of interesting and challenging problems. For the rest of this paper we are concerned with the opposite enhancement to communication: allowing the *user* to volunteer information.

Users volunteering advice. One of the major stumbling blocks in the successful use of knowledge-based systems is the general feeling of apathy with which many of these systems are met by the users. Much of the refusal to utilize systems such as MYCIN and REBOOTER stems from the fact that users, who often think of themselves as experts, feel that the system is telling them what to do. The system asks a question which the user answers. The system then decides, by some hidden mechanism, if it needs more information or is going to give the user advice. At no time are the users afforded the opportunity to make their observations known to the computer. They are simply allowed to answer the questions put to them -- a role which most humans do not experience as very satisfying. An expert who is knowledgeable about a domain wants to take an *active role* in the process of deciding what actions should be taken. While cooperative advice or criticism from a computer is welcome (e.g., like in the systems described above), the typical knowledge-based system that forces a particular format of discussion upon the user is not.

The GUS ("Genial Understanding System") system [Bobrow et al. 77]attempted to model a natural dialogue and it could cope with volunteered information. This was achieved by selecting a narrow domain (assisting the user in planning a trip) which constrained the range of expectations that GUS needed to have about the user's plans. The system was driven by a number of frames which characterized the domain and the dialogue itself.

Our contribution to increase the naturalness of communication and to eliminate some of the inflexibility is the introduction of mechanisms which allow the user to volunteer information to the system. We will first describe REBOOTER, an conventional knowledge-based system which we have built, used and evaluated. The shortcomings of REBOOTER led to the development of the SYSTEMS ASSISTANT, which is an illustration that enhanced communication capabilities are of crucial importance for knowledge-based systems.

3. REBOOTER: a Knowledge-Based System to Reboot Computers

Problem Description. REBOOTER is a knowledge-based system which allows users to reboot a PYRAMID 90X computer after it has crashed. It has a set of predetermined tasks which drive it to ask for certain pertinent information. If the user goal is to reboot the machine, REBOOTER first tries to get the machine running. When a certain state has been reached, the system will instantiate the task to boot the machine into single-user mode, and finally into multi-user mode. This process consists of five major tasks which are the initial status check (is the power on and can you log in), error recording, booting, file-system checking, and bringing the machine into multi-user mode. Examples of these tasks are in Figure 3-1 and a sample session with REBOOTER is described in Figure 3-2.

- **Status Query Task:** This task starts the REBOOTER by asking about the power and login status of the machine.
- **Reseat_boards Task:** This task may be initiated when there has been a problem rebooting the machine, the machine is up and a network problem has been found, or the REBOOTER suspects that a problem may be caused by a board being misaligned on the bus. Making sure the boards in the machine are seated properly often solves these problems.
- **Diagnose_noboot Task:** There are indications that there is an error in the booting process and further steps will be necessary to bring the machine back up (it is inside the *Diagnose_Noboot* task where the most sophisticated rules reside).

Figure 3-1: Task Examples

The initial status, error recording, file-system checking, and multi-user tasks are rule sets that ask basic questions (e.g., are there any error messages on the console) or require simple actions (e.g., please record any error messages in the log book). Inside the booting task are a number of sub-tasks. This is where the interesting rules reside and the data-driven paradigm is put to the test. It is inside this task where diagnosis of failed reboot attempts is carried out. The rules here help users determine what causes this failure. While automatic rebooting options are available, they are not able to deal with problems like hardware failures and serious file system errors. In these cases the machine will fail its attempts at reboot or will simply tell the operator that file system checking must be done manually. Unfortunately, experience shows that these conditions occur more often than we would like. Rebooting a computer, especially if the person is not totally familiar with it, is a non-trivial problem. This is demonstrated by the fact that 2 to 6 months of on the job training are done by our novice systems administrators before they are confident enough to reboot machines on their own. A computer is a complex and expensive piece of equipment which requires a lot of intuitive knowledge to deal with on an administrative basis. The rebooting process ranges from the trivial pressing of a couple of keys on the console to the complicated task of diagnosing hardware failures. REBOOTER, designed specifically to help with this process, can significantly reduce the complexity of this task. At the same time, it allows users to slowly incorporate this intuitive knowledge into their own knowledge structures by making them familiar with the types of actions necessary to perform this task.

Through our experience in rebooting computers, REBOOTER's contribution to the work of a novice systems employee is obvious. Novices simply do not know how to accomplish this task. They need to communicate with an expert to achieve their goal. Similarly, while experts can usually deal with rebooting problems, they often seek advice from other experts to confirm or enhance their understanding of those problems. Just as another pair of eyes can often uncover hidden bugs in a program, communication during the diagnosis of a reboot can often yield more useful plans of action. REBOOTER helps fill this role.

The Knowledge Base. The knowledge base of the REBOOTER, which excludes the user interface, consists of a set of OPS5 production rules [Brownston et al. 85]. The inference mechanism used is forward chaining which leads the structure of the rules to be in a task based, data-driven paradigm. The rules themselves decide when it is appropriate to switch from one task to another. Tasks are instantiated based on what the previous tasks were able to find out or accomplish. The program has two main modules, domain knowledge and explanation. Each module consists of several tasks, some of which are listed in Figure 3-1 for the domain module. Tasks consist of several rules related through the domain knowledge they analyze, and they comprise a question and answer session that guide both the user and REBOOTER through the problem space. A limited explanation module performs a post-analysis on the working memory elements left by the session and outputs its results, in the form of *canned text*, to a file which the user can then consult.

Communication Capabilities of REBOOTER. REBOOTER's user interface is a text based dialogue session that runs on traditional CRT terminals. REBOOTER presents a series of questions that lead the user through the five major tasks necessary to reboot the computer. As the dialogue session progresses, REBOOTER's knowledge base evolves through states which fire the necessary tasks in each of these five categories. A typical session with REBOOTER that represents a trouble-free reboot is reproduced below (see Figure 3-2).

This represents a system with traditional communication capabilities. Users are only allowed to answer questions that are put to them by REBOOTER. The system-driven dialogue session keeps the user in a passive role with respect to decision making in the reboot process. A graphical description of this type of communication is in Figure 3-3.

Shortcomings of REBOOTER. REBOOTER was put into use by the systems staff for a short period of time during which shortcomings in its design became apparent. Observations and discussions with users of the system yielded interesting results. While novice users are quite comfortable with the system-driven dialogue paradigm, expert users are quite irritated by it. In fact, expert users refused to use the system after their first or second experience with it. Discussions with the various users clearly indicated that experts do not want to be forced into a particular format of discussion with a system, while novices gain confidence in their actions through this very same mechanism.

Similar reactions were observed when the MYCIN [Buchanan, Shortliffe 84] program was introduced into the medical establishment. When experts in a field use a knowledge-based

```
INITIAL QUERY: Is the machine's power turned on?
<<yes/no>> yes
Can you log on to the machine that is down
across the network? <<yes/no>> no
Can you log on to the machine that is down at
the console? <<yes/no>> no

ERROR RECORDING: Record any error messages that
appear on the console. Go to the E_Frame by
pressing HOME then E. Are there any flashing
error codes? <<yes/no>> no

BOOTING: Go to the System Configuration Frame
(Frame 1) by pressing HOME then 1.  Press b
then z to boot the machine and start the CPU.
Do the two windows at the bottom right corner
of the console report that the machine is
BOOTED? <<yes/no>> yes
that the CPU is RUNNING? <<yes/no>> yes

SINGLE USER MODE: ......

SET THE DATE: ......

FILE SYSTEM CHECK: ......

MULTI USER MODE: To go into multi_user mode press
^D (control_D)

Execution halted by rule: multi_user_mode.
Would you like an explanation of the session?
<<yes/no>> yes

IF YOU GENERATED AN EXPLANATION IT WILL BE
FOUND IN:
/staff/system/stevens/rebooter/RULETRACE
THANKS FOR USING REBOOTER. MAIL ANY COMMENTS
TO CURT.
```

Figure 3-2: A Partial Session with REBOOTER

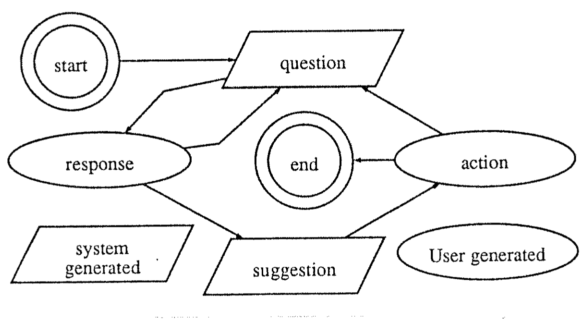


Figure 3-3: Control Flow in a System-Driven Dialogue

system they need to feel that they have an active role in the process of deciding what actions should be taken. In REBOOTER, the dialogue is completely system-driven. Users are delegated the tasks of answering questions and pushing buttons. MYCIN has the very same problem. Users are put in a passive role throughout their interaction with the system.

In the real world there are many instances of systems which, if implemented, *must* exhibit the property that users can immediately focus the attention of the system. For example, take a system that serves as an auto-pilot for an aircraft. If pilots observe something that involves an implied time constraint, they must have the ability to communicate this information to the system. Without this flexibility the system can never be used.

4. The SYSTEM'S ASSISTANT: Incorporating Information Volunteering

Our solution to this problem of inflexibility in the communication paradigm is the introduction of a mechanism through which the user can volunteer information to the system. By volunteering information we mean that the user can make statements about the domain which are out of context with respect to the current conversation between user and system. Information volunteering allows users to be in the speaker role and focus the attention of the system on the information which they feel is relevant. The user is no longer just answering questions, but taking an active role in deciding what the knowledge-based system is reasoning about. The system now plays the role of assisting users as opposed to directing users and therefore this new version of our knowledge-based system is called the SYSTEMS ASSISTANT (the term SYSTEMS ASSISTANT is derived from the name of the group which maintains the computers in the Computer Science department. The group is called the *Systems Group*, hence the name SYSTEMS ASSISTANT) Information volunteering is probably best explained by way of an example:

> *When a user first starts up a session with the SYSTEMS ASSISTANT the system will always begin by asking some basic information about the PYRAMID in question. This information must be known to the* SYSTEMS ASSISTANT *for it to do any diagnosis or offer any assistance. Beyond that point, however, the actions which the* SYSTEMS ASSISTANT *will take are mostly dependent upon the data which the user supplies in response to its inquiries. The* SYSTEMS ASSISTANT *asks a question after which the user responds with some new data. After reviewing the modified state of the data at hand the* SYSTEMS ASSISTANT *proceeds to suggest some course of action which is then carried out by the user. This loop (see Figure 3-3) continues until the* SYSTEMS ASSISTANT *has successfully helped the user reboot the computer. However, an experienced systems administrator will be able to notice pertinent information long before the* SYSTEMS ASSISTANT *asks about it. For instance, these types of users might quickly notice that the ethernet board is sticking an inch further out than the rest of the boards in the machine. They would certainly come to the conclusion that this might have something to do with the machine's problem, and therefore want to focus the attention of the* SYSTEMS ASSISTANT *on that fact. This type of information is considered out of context since the*

SYSTEMS ASSISTANT *is asking questions like IS THE MACHINE'S POWER TURNED ON, or DOES THE CONSOLE SAY THAT THE CPU IS RUNNING. If users know something about the system, then they should be able to present that information to the* SYSTEMS ASSISTANT *as soon as it becomes apparent.*

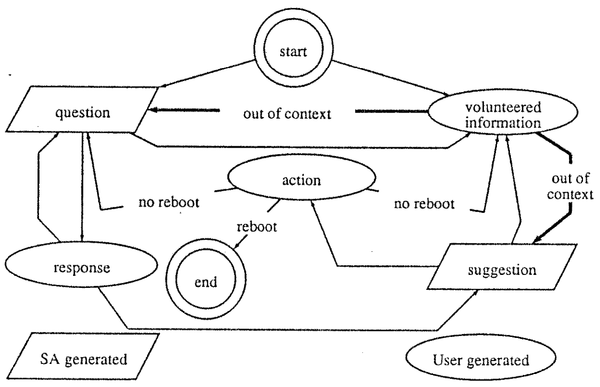
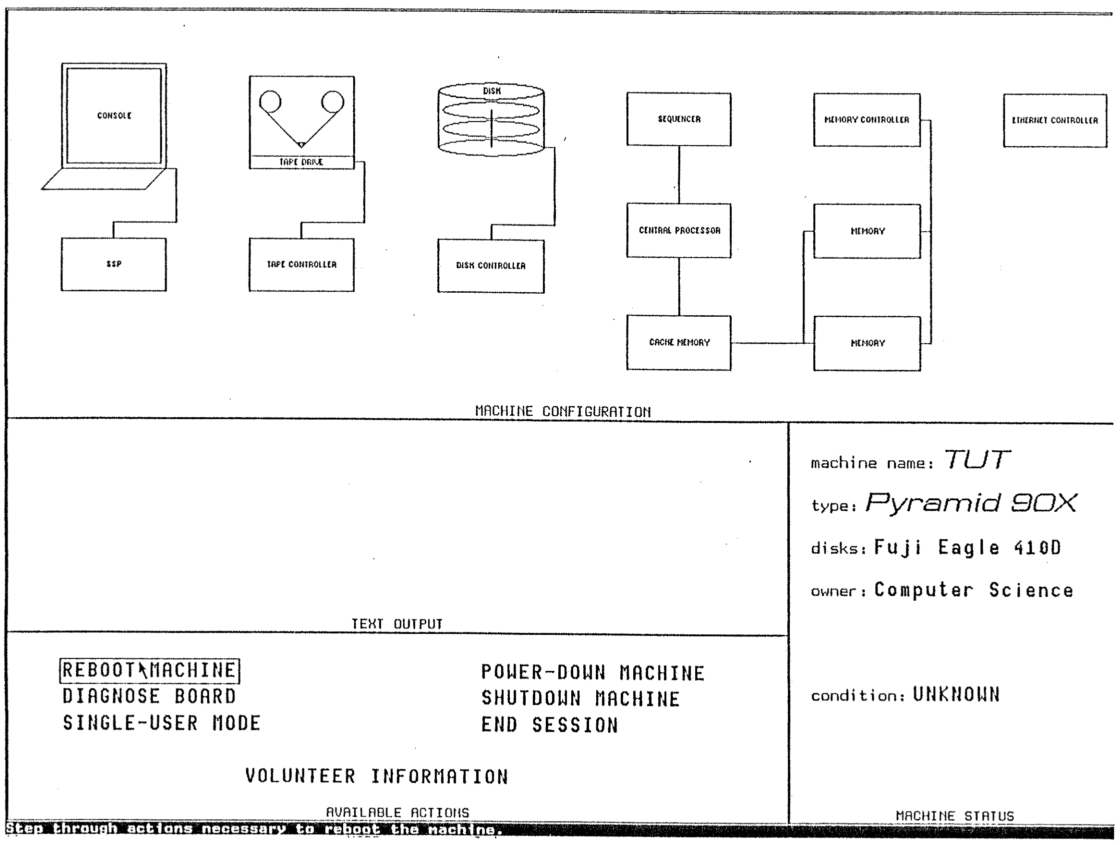


Figure 4-1: Control Flow in a Mixed-Initiative Dialogue

The SYSTEMS ASSISTANT requires an extended model of interaction (see Figure 4-1), incorporates a new interface (see Figure 4-2), and requires a major restructuring of the knowledge base used in REBOOTER.

The system's knowledge is explicitly represented in a world model with which the user interacts in a direct manipulation style [Hutchins, Hollan, Norman 86]. The different hardware components of the PYRAMID are represented in graphical form (see Figure 4-2).

Users can either ask for general information about each of these components or volunteer information about them. If users are confused about what an icon represents they can ask the system about that icon by clicking the mouse on it. At that point the system presents the user with a text based explanation about the component in question. It also explains some of the most common indications that this component is damaged and common methods of determining the functional state of it. These possibilities give the user a window into the "mind" of the SYSTEMS ASSISTANT and provide a well defined and common basis for communication between the user and the system. To volunteer information (and change the context in which the icons are understood), the users click with the mouse on the volunteer information icon. At this point a click on any of the machine component icons yields a menu of possible facts about that particular component. Since we are asking the user to volunteer information that is best described by natural language, but are not able to allow the actual use of natural lan-



machine name: TUT

type: Pyramid 90X

disks: Fuji Eagle 410D

owner: Computer Science

condition: UNKNOWN

Step through actions necessary to reboot the machine.

Figure 4-2: Initial State Of The SYSTEMS ASSISTANT

guage, we present the user with a menu of text based choices. This menu defines for the user the possible space of information which can be understood by the rule base of the SYSTEMS ASSISTANT. Figure 4-3 is a typical example of what one of these menus looks like. On the left side are the common problems associated with this particular piece of hardware. On the right side are the choices which indicate that one of the problem areas has already been checked, and at the bottom is a choice indicating an unfounded suspicion that something has gone wrong with that piece of hardware. In this manner the user is afforded the opportunity to volunteer out of context information.

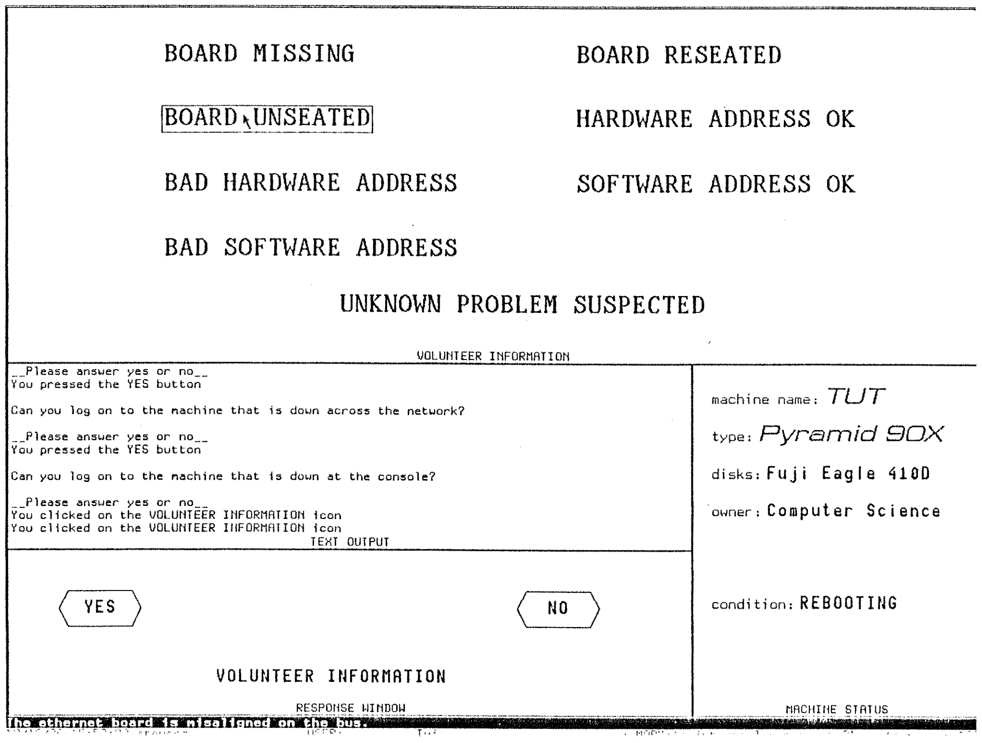
The interface, however, is not the most crucial modification that is necessary. To bring information volunteering to fruition it is not sufficient to change the external appearance of the system on the screen. This new mechanism requires the restructuring of the knowledge base to accommodate the incoming out of context information. In REBOOTER, an analysis of the structure of tasks was carried out to determine which task should in turn instantiate successive tasks. The original design was far too rigid for the information volunteering mechanism. What is needed is a more general methodology for determining the current task selection. This problem is being solved by removing the task selection criterion from the tasks themselves and creating an autonomous collection of rules whose only function is to recognize situations in which particular tasks should be instantiated. To operate in this mode the system needs more information about the machine components and its own rule groups than before to allow the SYSTEMS ASSISTANT to resolve conflicts when more than one task is simultaneously instantiated due to some volunteered information. This extra knowledge allows the SYSTEMS ASSISTANT to be much more powerful in its ability to handle the inevitable context switches that occur due to the incoming out of context information.

In addition, a mechanism is needed through which the system can determine what information is implicit in the volunteered information. For instance, this instantiation of tasks might be altered if users volunteer information that implies they have already tried to reboot the machine. A related problem is the decision of whether to ask a previously posed question again. The volunteered information might have implied an answer to this earlier question and the rule base has to be general enough to handle these cases.

5. Experiences and Future Research

The shortcomings of the REBOOTER clearly indicated that knowledge-based systems will not be accepted if their communication capabilities are too limited. The design and the implementation of the SYSTEMS ASSISTANT provided another piece of evidence (along the findings of the DIPMETER system [Smith 84]) that designing the knowledge base and the inference engine of a knowledge-based system may be a much easier task than providing these systems with the right kind of communication capabilities. Making a system able to accept volunteered information is not just a matter of redesigning the interface to that system but requires that the knowledge base be expanded and reorganized.

The SYSTEMS ASSISTANT seems to provide the right kind of mixture between highly structured dialogues (which are useful for the novice) and the possibility to volunteer information to get to the point quickly, which is a necessary requirement to make a



```
__Please answer yes or no__
You pressed the YES button

Can you log on to the machine that is down across the network?

__Please answer yes or no__
You pressed the YES button

Can you log on to the machine that is down at the console?

__Please answer yes or no__
You clicked on the VOLUNTEER INFORMATION icon
You clicked on the VOLUNTEER INFORMATION icon
```

machine name: TUT
type: Pyramid 90X
disks: Fuji Eagle 410D
owner: Computer Science
condition: REBOOTING

```
The ethernet board is misaligned on the bus.
```

Figure 4-3: Volunteering Information About The Ethernet Controller

system acceptable to the expert.

Many more features should (and will be) added to the SYSTEMS ASSISTANT. Having a sensory system (which signals the state of the broken machine) connected to the SYSTEMS ASSISTANT would allow it to monitor the actions taken by the user. Users should also be able to query the system on *how or why* it does anything. If the system says that the ethernet board needs to be reseated on the bus, users might want to know how to do this, or why the system feels that this is necessary (the second question requires more elaborate explanation capabilities than most system currently have).

An extension in another dimension, which is closely related to our work supporting human problem domain communication [FischerLemke1987a], is to allow users of the system to create their own machine configurations with the assistance of construction and design kits.

If users are not willing to use the systems we design, a major component of the employed theory and methodology must be missing. In many cases this resistance will be based on the limited communication capabilities. Natural communication is a crucial aspect to increase the usefulness and usability of computers. Information volunteering is an important part of it which should be explored in other task domains. Knowledge-based systems of all types can benefit from allowing the user to have more control.

Acknowledgements

The authors would like to thank William Pachoud who participated in the development of the REBOOTER and the SYSTEMS ASSISTANT and the systems staff of the Computer Science department who used the REBOOTER and showed us its shortcomings.

This research was supported by: grant No. DCR-8420944 from the National Science Foundation, grant No. MDA903-86-C0143 from the Army Research Institute and a grant from Microelectronics and Computer Technology Corporation (MCC), Austin.

References

[Bates, Bobrow 84]
M. Bates, R.J. Bobrow, *Natural Language Interfaces: What's Here, What's Coming, and Who Needs It*, in W. Reitman (ed.), *Artificial Intelligence Applications for Business*, Ablex Publishing Corporation, Norwood, NJ, 1984, pp. 179-194, ch. 10.

[Bobrow et al. 77]
D.G. Bobrow, R.M. Kaplan, M. Kay, D.A. Norman, H. Thompson, T. Winograd, *GUS, A Frame-Driven Dialog System*, Artificial Intelligence, No. 8, 1977, pp. 155-173.

[Bolt 84] R.A. Bolt, *The Human Interface*, Lifetime Learning Publications, Belmont, CA, 1984.

[Brownston et al. 85]
L. Brownston, R. Farrell, E. Kant, N. Martin, *Programming Expert Systems in OPS5: An Intoduction to Rule-Based Programming*, Addison-Wesley Publishing Company, Reading, MA, 1985.

[Buchanan, Shortliffe 84]
B.G. Buchanan, E.H. Shortliffe, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, Addison-Wesley Publishing Company, Reading, MA, 1984.

[Carbonell 70]
J.G. Carbonell, *AI in CAI: An Artificial-Intelligence Approach to Computer-Assisted Instruction*, IEEE Transactions on Man-Machine Systems, Vol. MMS-11, No. 4, December 1970.

[Fischer 83]
G. Fischer, *Symbiotic, Knowledge-Based Computer Support Systems*, Automatica, Vol. 19, No. 6, November 1983, pp. 627-637.

[Fischer 86]
G. Fischer, *Cognitive Science: Information Processing in Humans and Computers*, in H. Winter (ed.), *Artificial Intelligence and Man-Machine Systems*, Springer-Verlag, Berlin - Heidelberg - New York, 1986, pp. 84-112.

[Fischer 87]
G. Fischer, *A Critic for LISP*, Proceedings of the 10th International Joint Conference on Artificial Intelligence (Milan), 1987.

[Fischer, Lemke 87]
G. Fischer, A.C. Lemke, *Constrained Design Processes: Steps Towards Convivial Computing*, in R. Guindon (ed.), *Cognitive Science and its Application for Human-Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[Fischer, Lemke, Schwab 85]
G. Fischer, A.C. Lemke, T. Schwab, *Knowledge-Based Help Systems*, Human Factors in Computing Systems, CHI'85 Conference Proceedings (San Francisco, CA), ACM, New York, April 1985, pp. 161-167.

[Hutchins, Hollan, Norman 86]
E.L. Hutchins, J.D. Hollan, D.A. Norman, *Direct Manipulation Interfaces*, in D.A. Norman, S.W. Draper (eds.), *User Centered System Design, New Perspectives on Human-Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1986, pp. 87-124, ch. 5.

[Smith 84]
R.G. Smith, *On the Development of Commercial Expert Systems*, AI Magazine, Vol. 5, No. 3, Fall 1984, pp. 61-73.