

COMMUNICATION REQUIREMENTS FOR COOPERATIVE PROBLEM SOLVING SYSTEMS

GERHARD FISCHER

Department of Computer Science and Institute of Cognitive Science, University of Colorado, Boulder,
CO 80309, U.S.A.

(Received for publication 18 October 1989)

Abstract—Despite lip service that “most knowledge-based systems are intended to be of assistance to human endeavor and are almost never intended to be autonomous agents”, knowledge-based systems research has not been focused enough on the nature and the requirements of cooperative problem solving systems.

The emphasis of our work is on creating computer systems to facilitate the cooperation between a human and a computer. Cooperation requires more from a system than having a nice user interface or supporting natural language dialogs. One needs a richer theory of problem solving, which analyzes the functions of shared representations, mixed-initiative dialogs, argumentation and management of trouble.

Our evolving theoretical framework for this approach has led to a number of prototypical systems developments which serve as vehicles for future research. Examination of these systems provides evidence that learning and effective problem solving can be improved through the use of cooperative problem solving systems.

Key words: Cooperative problem solving, expert systems, mixed initiative dialogs, shared knowledge, natural communication, speaker vs listener role, situation model vs system model, information access, information volunteering, design environments, HELGON, SYSTEMS' ASSISTANT, JANUS.

1. THE NEW LOOK OF ARTIFICIAL INTELLIGENCE

Our goal is to establish the conceptual foundations for using the computational power that is or will be available in computer systems towards the goal of creating cooperative problem solving systems. We believe that artificial intelligence methodologies and technologies provide the unique opportunity to improve productivity by addressing, rather than ignoring, human needs and potential. In the spirit of Einstein's remark “*My pencil is cleverer than I*”, we are building systems which *augment* and *amplify human intelligence* in problem solving, decision making and information management rather than replacing it.

Traditionally, the most widely understood goal of artificial intelligence has been to understand and build autonomous, intelligent, thinking machines. We believe with a number of other researchers that a more important goal is to understand and build interactive knowledge media [1] or cooperative problem solving systems [2-4].

Cooperative problem solving in our approach refers to the cooperation between a human and a computer (see Fig. 1). On the one hand, it shares a large number of research issues with two related research areas, namely *Computer Supported Cooperative Work (CSCW)* [5], which refers to cooperation between humans mediated by computer and *Distributed Artificial Intelligence* [6], which refers to a

cooperation between computer systems. On the other hand, it poses a number of unique challenging problems as they occur in a large number of joint human-computer systems [7].

This paper discusses the differences between traditional expert systems and cooperative problem solving systems, analyzes the special requirements of cooperative problem solving systems and defines a theoretical framework for these systems. It describes a number of prototypical system developments, each addressing a different aspect of cooperative problem solving. HELGON is an information access tool which supports cooperative problem solving by helping users to specify their goals incrementally. The SYSTEMS' ASSISTANT allows users to volunteer information in support of mixed-initiative dialogs. JANUS is a design environment which integrates constructive and argumentative design.

2. COOPERATIVE PROBLEM SOLVING SYSTEMS vs EXPERT SYSTEMS

The major difference between classical expert systems (such as MYCIN and R-1) and cooperative problem solving systems is that the human is much more an active agent and participant. Traditional expert systems asked the user many questions and then returned an answer. In a cooperative problem solving system the user and the system share the problem solving and decision making and different role distributions may be chosen depending on the user's

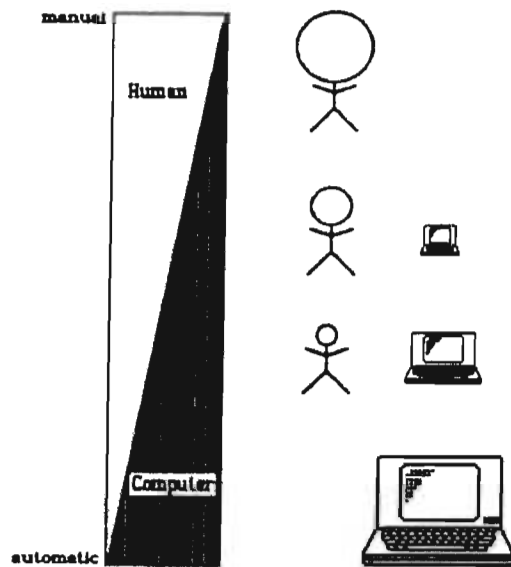


Fig. 1. Cooperative systems. To achieve a task, different systems' architecture are possible, ranging from manual (everything is done by the human) to completely automatic (everything is done by the computer). Cooperative systems explore different role distributions between the two opposing ends.

knowledge, the user's goals and the task domain. A cooperative system requires much richer communication facilities than the ones which were offered by traditional expert systems. It raises two important questions:

- What part of the responsibility still has to be exercised by human beings?
- How do we organize things so that the intelligent part of the automatic system can communicate effectively with the human part of the intelligent system?

The two types of systems can be characterized and differentiated along the following issues:

Partial understanding and knowledge of complex task domains—The interaction paradigms for dealing with complex information stores (e.g. high-functionality computer systems such as LISP machines containing tens of thousands of objects and tools [8]) have often been based on the unfounded assumption that people using these systems approach them with a precisely described task. But in most problem-solving and information retrieval tasks, the articulation of a precise task is the most difficult problem. Users of such systems suffer from a lack of knowledge about the interdependencies between problem articulation and specification, and of knowledge about the tools that exist for solving these problems. Ignorant of these mappings, users are unable to develop a complete specification of what they want; specifications must be constructed incrementally.

The communication requirements for these systems must allow that a question can be phrased in a variety of ways. Novices cannot ask questions about knowl-

edge that they do not know exists, and they may not be able to articulate their questions without the help of the expert. They ask many questions initially at a very general level, and a good deal of dialogue must occur before the communicators attain sufficient level of specificity.

The failure of autonomous approaches—Cooperative systems are based on a successful combination of human skills and computing power in carrying out a task which cannot be done either by the human or by the computer alone. We illustrate our conception of cooperative systems by giving examples in domains where autonomous systems have failed:

- *Computerized axial tomography* (CAT scanning [9]) is based on a cooperation between doctor and computer. The necessary inverse Fourier transformations involve an immense amount of computation and cannot be done without the help of a computer—and the interpretation of the data requires discrimination between subtle differences in density which is beyond current capabilities in image processing.
- Kay [10] proposes a *symbiotic machine translation system* that is always under the tight control of translators. The system is there to help increase their productivity and not to supplant them. The fully automatic approach has failed badly in the past.
- In *aircraft automation* [11] two different models are under investigation: the pilot's assistant and the electronic copilot which can be differentiated along the separation of tasks and control between humans and machine.

Two agents can achieve more than one—Cooperative problem solving systems consisting of a human and a computer can exploit the asymmetry of the two communication partners. Humans can contribute what they can do best (e.g. use of common sense, goal definition, decomposition into subproblems, etc.), whereas the computer should be used for what it is good for (e.g. external memory support, consistency maintenance, hiding irrelevant information, intelligent summarizing, visualization support, etc.).

Breakdowns in cooperative problem solving systems are not as detrimental as in expert systems—Effective assistance is a collaborative effort in which advisor and client work together to detect and repair troubles that arise.

One can never anticipate or "design away" all of the misunderstandings and problems that might arise during the use of these systems. We need to recognize and develop system resources for dealing with the unexpected. The problem is not that communicative trouble arises that does not arise in human-to-human communication, but rather than when these inevitable troubles do arise, there are not the same resources available for their detection and repair. A cooperative agent needs to understand the nature of open problems, the intentions of the problem solver,

and the fact that goals are often modified during a problem solving process.

Background assumptions do not need to be fully articulated—We need a better understanding of the possibilities and limitations of expert systems research. We have to define the characteristics for problems which are suitable for expert systems research to generate realistic expectations. When we talk of a human expert, we mean someone whose depth of understanding serves not only to solve specific well-formulated problems, but also to put them in a larger context. The nature of expertise consists not only in solving a problem or explaining the results (which some expert systems can do to some extent), but of learning incrementally and restructuring one's knowledge, of breaking rules, of determining the relevance of something and of degrading gracefully if a problem is not within the core of the expertise.

Semi-formal system architectures are appropriate—Semi-formal systems do not require that the computer can interpret all information structures, but it may serve only as a delivery system of information to be read and interpreted by the human. Semi-formal systems (which are also studied in computer-supported cooperative work) can be used more extensively in cooperative systems than in expert systems and will play a large role in the design of effective joint human-computer systems.

Humans enjoy "doing" and "deciding"—In many situations, humans enjoy the process—not just the product; they want to take part in something. This is why they build model trains, why they plan their vacation and why they design their own kitchen. Automation is a two-sided sword. At one extreme, it can be regarded as a servant, relieving humans of the tedium of low-level control operations, freeing them for higher cognitive functions. At the other extreme it is viewed as reducing the status of humans to "button pushers," and stripping work of its meaning and satisfaction.

3. REQUIREMENTS FOR COOPERATIVE PROBLEM SOLVING SYSTEMS

3.1. Beyond user interfaces

Effective human-computer communication is more than creating attractive displays on a computer screen: it requires providing the computer with a considerable body of knowledge about the world, about users and about communication processes. This is not to say that the user interface is not of crucial importance to knowledge-based systems. Analysis of expert systems, e.g. of the DIPMETER advisor [12], has shown that the acceptance and real use of expert systems depends on far more than a knowledge base and an inference engine. The developers examined the relative amount of code devoted to different functions of DIPMETER and found

that the *user interface portion was 42%* compared to 8% for the inference engine and 22% for the knowledge base. Similar data is reported for commercial knowledge-based system tools (e.g. in Intellicorp's tools, 55–60% of the code is interface related [13]). A good user interface is important for two groups: for the developers of knowledge-based systems and for the end-user of these systems.

The communication requirements are even more important for cooperative problem solving systems. Because the user is actively involved in the problem solving and decision making process, there is an increased necessity for the interface to support the task at a level that is comprehensible by the user. In order for a knowledge-based system to support cooperative problem solving, the following components dependent critically on each other:

- The structure of the knowledge and problem solving system itself—how does a system represent its problem solving activity and retrieves the relevant portion appropriately in response to user queries.
- The generation of views of this knowledge which corresponds to the needs and the knowledge of the user; in order to do so, a system must contain a model of the user.
- The external presentation of this knowledge on the screen; it is this part which is mostly (explicitly or implicitly) associated with user interface research.

3.2. Mixed-initiative dialogs

Despite the fact that communication capabilities such as *mixed-initiative dialogs* [14] have been found to be crucial for intelligent systems, the progress to achieve them has been rather modest.

One model frequently used in human-computer systems (e.g. MYCIN) is the *consultation model*. From an engineering point of view, it has the advantage of being clear and simple: the program controls the dialog (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. Mixed-initiative dialogs must support information volunteering by the system as well as by the user.

Information volunteering by the user—Real users of expert systems are not data entry clerks. 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. The SYSTEMS' ASSISTANT (see Section 5.2) is a system which allows the user to volunteer information [42].

Information volunteering by the system—Humans often learn by receiving answers to questions which they have never posed or which they were unable to pose. 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 [15] and critic systems (see Section 5.3)), 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 to help and advice.

3.3. Knowledge requirements

Shared understanding—To increase the mutual intelligibility between agents in cooperative problem solving requires a deeper understanding of the reciprocal recognizability of plans, enabled by common conventions for the expression of intent, and shared knowledge about typical situations and appropriate actions [16]. For example, taking aviation as an example [17]: the notion that human operators should inform the system of their intentions, or goals, may seem simple, but it is a capability noticeably and perhaps dangerously lacking in most present-day automatic systems. Goal sharing (also called “intent-driven systems”) would first require that the crew make its intentions known (“here’s what we want to do”), and then allow the computer to check crew inputs and system outputs to see if they are logically consistent with the overall (strategic) plan. The automation (e.g. the digital flight guidance system or long-range navigators) uncritically accepts inputs and has no capability for checking their overall consistency with any understood goal (a capability such as this might have saved KAL Flight 0007). Likewise, goal sharing may have prevented some of the dramatic fuel incidences that have occurred in recent years.

The relevance of human problem-domain communication—Many current knowledge-based systems use knowledge representations at a too low level of abstraction. This makes both system design and explanation difficult, since the system designer has to transform the problem into a low-level implementation language and explanation requires translating back to the problem level. Cooperative problem solving systems must have knowledge about task domains. To create powerful tools for the humans, we must “teach” the computer the languages of application domains. Systems with abstract operations and objects of a domain built into them give the impression of *human problem-domain communication* [18] rather than human-computer communication. Human problem-domain communication reduces the cognitive transformation

distance between problem-oriented and system-oriented descriptions [19].

3.4. Beyond natural language: natural communication

Natural communication is more than the ability to communicate in natural language. It is the ability to engage in a dialog 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.

Responsible proponents of natural language interaction acknowledge that current programs cannot understand language in a significant sense [20]. This does not rule out the use of natural language interfaces because many practical applications (e.g. access to databases) do not demand deep understanding. The practicality of limited natural language systems is still an open question. Since the nature of the queries is limited by the formal structure of the data base, it may well be more efficient for a person to learn a specialized formal language designed for that purpose, rather than learning through experience just which natural language sentences are and are not accepted. The *habitability* [21] of a system (which measures how quickly and comfortably a user can recognize and adapt to the system’s limitation) is a critical issue which needs to be studied empirically in realistic situations.

These dynamics of cooperative problem solving indicate why *natural language interfaces* to databases do not solve the critical problem of access to information. Studies (e.g. [4]) have provided evidence that a natural language front-end is a fallacy. Current natural language interfaces support the translation of a fully articulated query from natural language into a formal query language, but they do not assist users who are unable to describe precisely what they want at the beginning of an information-seeking process.

Based on the fact that there exists an *asymmetry* between human and computer, the design of the interface is not only a problem of simulating human-to-human communication but of engineering alternatives to interaction related properties. We do not have to use natural language for every application; some researchers claim that, in many cases, it is not the preferred mode of communication [21, 22]. In natural language interfaces, the computer is the listener and the human the speaker. The listener’s role is always more difficult, because he/she has to understand a problem based on the speaker’s description. Our work has been primarily guided by the belief, that it is the user that is more intelligent and can be directed into a particular context. This implies that the essence of user-interface design is to provide users with

appropriate cues. Windows, menus, spreadsheets and so on provide a context (making the machine the speaker and the human the listener) that allows the user's intelligence to keep choosing the next step.

3.5. Models of the communication partner

Cooperative problem solving systems require that users have models of the systems with which they interact and systems have models of their users. The later model is needed to provide *explanations* and *differential descriptions*. Explanations need to be given at the right level of detail and with the right level of assumed shared understanding about the other agent's knowledge. Differential descriptions (describing something by relying on the basic paradigm "x is equal to y, except. . .") is a powerful paradigm (demonstrated in the programming world with object-oriented inheritance hierarchies).

3.6. Beyond "one-shot" affairs

Even with good models of users, cooperative problem solving systems cannot be restricted to "one-shot" affairs. One cannot always be right the first time and one cannot guarantee that an advice or criticism is understood. The HELGON system (see Section 5.1) is built on the basic assumption that users do not always have well-formulated goals to start with, but that these goals are constructed incrementally in a cooperative fashion.

Cooperative problem solving systems support the incremental construction of queries and goals, whereas expert systems require a complete specification in order that they are able to solve a problem.

3.7. Empirical studies

A careful assessment of the systems that we have built over the last decade (e.g. critics [23] and active and passive help system [15]) has shown that they fall short of supporting real cooperative problem solving. In order to obtain a deeper understanding of cooperative problem solving, we have designed and carried out an empirical study of a high-functionality system, namely the McGuckin hardware store in Boulder (which offers more than 300,000 items) and the cooperative problem solving between customers and sales agents. Details of these studies are contained in [24]—the most important findings are summarized here briefly.

- *Incremental query specification*—Frequently customers did not know what they really needed and did not know how their problems could be mapped onto the items which the store offers. Their queries were constructed *incrementally* through a cooperative problem solving process between a customer and a sales agent.
- *From natural language to natural communication*—People rarely spoke in complete, grammatical sentences, yet managed to communicate in a natural way. This observation clearly indi-

cates that the support of natural communication (which allows for breakdowns, follow-up questions, clarifying dialogues, explanations, etc.) is much more important than being able to parse complex syntactic sentences.

- *Mixed-initiative dialogs*—People are flexible in the roles they play during a problem solving episode. They easily switch from asking to explaining, from learning to teaching. The structure of these dialogs were neither determined by the customer nor by the sales agent, but clearly indicated mixed initiatives [25] determined by the specifics of the joint problem solving effort.
- *Multiple specification techniques*—We observed a great variety of different specification techniques ranging from bringing in a broken part to very general request such as "I need a lock for my doors that reduces my insurance rate".
- *Management of trouble*—Many breakdowns and misunderstandings occurred during the observed problem solving episodes. But in almost all cases, clarifying dialogs allowed their recovery, illustrating the important feature that problem solving among humans cannot be characterized by the absence of trouble, but by the identification and repair of breakdowns (the major contribution of the research work of [16] was to clearly identify this issue).
- *User modeling*—The study made it evident that user modeling plays a crucial role in identifying the right level of shared understanding and providing the appropriate level of explanation and help.

The state of the art with respect to complex computer systems can be characterized by saying "High-functionality computer systems offer the same broad functionality as large hardware stores, but they are operated like department stores", i.e. what is missing for them is the cooperative support of knowledgeable sales agents.

4. A THEORETICAL FRAMEWORK FOR COOPERATIVE PROBLEM SOLVING SYSTEMS

4.1. A taxonomy of cooperative problem solving systems

Humans and computers play different roles in cooperative problem solving processes. Within a cooperative problem solving system, the computer can play different roles. It can be a tutor [26], a suggestor [18], an advisor [27] or a critic [28, 29, 43].

In our work we have concentrated on critics [23]. *Computer-based critics* are a paradigm for intelligent human-computer communication which overcomes a number of limitations of other approaches (such as tutoring and advising). Critics are much more user-centered and support users in their *own* doing. They allow users to do whatever they want and interrupt

only when the user's plans, actions or products are considered significantly inferior. They are applicable in situations where users have some basic competence in carrying out a task, because users must be able to generate a plan, action or product by themselves. They are most useful in domains where no unique best solution exists but where trade-offs have to be carefully balanced.

4.2. Speaker vs 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 [30]. Natural language should not be used for every application; in many cases it is not the preferred mode of communication [21].

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, 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 based on a world-model of computation rather than on a conversational model [19].

The different role distributions are illustrated in the context of the HELGON system (see Section 5.1). Specifying retrieval cues is often difficult because users don't know exactly the term which is to be used. Using examples as cues is one way in which information can be retrieved. But if the users know exactly what they want, they can ask for it directly, and there is no need to bother with examples. HELGON thus supports two modes of operation: users can be in the

listener role, learning from the system what sort of questions they can ask, or in the speaker role, asking the system for what they want. Figure 2 lists some of the advantages and disadvantages associated with either role.

4.3. Situation vs system model

The *situation model* [31] is a mental representation of the situation as the user sees it, including the problems motivating a task, general ideas for finding a solution, and a characterization of the desired goal state. The *system model* consists of a set of operations that, when invoked, will result in the desired solution. These operations must all be within the repertory of the system; that is, for each operation there must exist one or more commands, depending upon context, to execute it. At the level of the situation model, goals refer to actions and states in the user's problem space and are articulated in terms of what we want. Goals may be precise or imprecise, but the important point is that they are not necessarily structured or named according to the system. It is subjective and varies somewhat among individuals.

In order to get something done on a computer system, however, the user's situation model must be transformed into a system model, which is normative and system specific. Our question has been, how, for a variety of tasks in which information management plays a central role this transformation from situation model to system model is achieved, and what system support can be provided for it.

In Fig. 3 several different approaches to bridging the gap between the situation and system model are outlined:

- (1) illustrates the normal situation, where there is no support to bridging this gap. In this case, people frequently have difficulties in solving problems or finding information, because they are unable to generate an effective system model, even if they have a clear understanding of the situation involved.
- In (2), a new system model is constructed which is closer to an individual's situation model and hence easier to understand [44].
- In (3) (which represents the HELGON approach [32]), we attempt to let users bring their situation model closer to the system model by making the relevant features of the latter more transparent: e.g. in HELGON information retrieval starts with

User in listener role:

- *Specification of information:* Clicking at information displayed on the screen.
- *Advantage:* Only terms that the system knows can be used.
- *Disadvantage:* The information has to be on the screen.

User in speaker role:

- *Specification of information:* Keyboard input.
- *Advantage:* Users can type in values they know right from the beginning.
- *Disadvantage:* Users may use terms the system does not know.

Fig. 2. Speaker vs listener role.










Situation Model	System Model	"normal"
		no intelligent support system
 restructuring		NewScope, InfoScope
reformulation 		Helgon, Retrieve
  agent 		Network
 <u>training</u> 		CS course, tutor, etc.

Fig. 3. Situation vs system model.

a query which is formulated at the level of the situation model, but is then incrementally reformulated into the system model. How an initial query stated in situation model terms is transformed into a query the system can actually understand in HELGON is illustrated in Section 5.1.

- In (4), a knowledge-based agent translates a request in the situation model into the system model [45].
- Finally, (5) is another normal case, where users are trained to express themselves in the system model. This is the expert case, where the user's situation model already takes into account the constraints of the system.

5. PROTOTYPICAL SYSTEM DEVELOPMENTS TOWARDS COOPERATIVE PROBLEM SOLVING SYSTEMS

5.1. HELGON: new approaches to information access

People who attempt to use a complex information store on a computer encounter a number of problems. They do not know what information exists or how to find information, they get no support in articulating a question, and they are unable to phrase their question in terms that the system understands. HELGON, an intelligent environment that supports limited cooperative problem solving, helps people deal with complex information stores. HELGON supports retrieval and editing by reformulation with multiple specification techniques, and it acquaints the user with the system model of the information store. HELGON can also be seen as a new tool for *knowledge utilization* which also knowledge editing.

5.1.1. *The conceptual framework behind HELGON. Retrieval by reformulation*—HELGON [32] is based on the paradigm of retrieval by reformulation [33],

which was derived from a theory of human remembering. This theory postulates that people naturally think about categories of things not in terms of formal attributes but in terms of examples. HELGON supports the incremental description of a desired object with multiple specification techniques. Systems that support retrieval by reformulation are cooperative in the sense that after users give an initial description, the system responds with an indication of its understanding by displaying example items from the knowledge base that match this description. Users then refine their description based on this feedback until a suitable item is found or until the absence of such an item is established.

Situation model and system model in HELGON—Many current information stores (e.g. help systems) are oriented toward the system rather than toward the user. That is, information is structured around a description of the system, not around an analysis of the problems users address when using the system. For example, a shortcoming of many existing information stores is that access is by *implementation unit* (e.g. LISP function, UNIX command) rather than by *application goal* on the task level.

The successful use of a complex information store requires that the goals expressed in the situation model be translated in terms of the system model (see Fig. 4). HELGON's approach to solving this problem is to make the system model more obvious to the user by providing support tools for easy access to the "world" of the system.

5.1.2. *Retrieval by reformulation in HELGON.* The query is initialized with the root node of the category hierarchy. The list of items matching the query is shown in the **Matching Items** pane, and one of the matching items is shown in the **Example of the Matching Items** pane (see Fig. 5). The query consists of categories and attribute restrictions associated with the categories. Categories as well as attribute values can be either "required" or "prohibited." The user does this by selecting them from the screen or a menu of alternative values or by typing them in on the keyboard. When the user makes additions to the query through input on the keyboard, only useful values, that is, values that exist in the knowledge base, are accepted. This prevents the user from imposing a restriction that would by itself lead to no matching items (e.g. because of a typographical error). The system gives help by completing partial input automatically if it is unique or by listing all possibilities that contain the current input as a substring.

Users can create the query top-down by selecting from the category hierarchy display the category that is expected to contain the desired information. But users may not know in what category the information is stored. Therefore, they can also work bottom-up by criticizing the example. A problem of this approach is that, in a large information space, the example given might be too far away from the desired

Query as represented in a specific user's situation model:

"Find the reference for the final ONR project report from CU."

A corresponding query in system-understandable terms:

TECHREPORT
 INSTITUTION: Department of Computer Science,
 University of Colorado
 YEAR: 1988

Fig. 4. Situation model vs system model.

information. Multiple specification techniques, e.g. first narrowing down the information space by selecting categories, then continuing by criticizing examples, are therefore important in dealing with complex information stores.

Visualization of the information store—It is well-known that users become disoriented in large information stores [34]. HELGON allows therefore to display the structure of the underlying information store, i.e. a hierarchy of categories, graphically. Once users have found a category that seems likely to hold the information they are looking for, they can add it to the query with a mouse click. They can also use the graphical display to edit the underlying structure of the information store (e.g. new subcategories can be created).

Browsing—In addition to assisting the user in defining a query, HELGON supports browsing in the information store. The graphical category hierarchy display can be used to browse categories. Links

within the information units can be followed, that is, items that appear as attribute values of other items (displayed in bold face) can be inspected. And items that users looked up previously are added to the **Bookmarks of Items** and allow users to return easily to previous states of their information search.

Editing by reformulation—HELGON is not just a tool for *viewing* information—one of the shortcomings we identified with the RABBIT and ARGON systems. It allows users to *edit* information and integrates the creation of new knowledge base items with the retrieval by reformulation paradigm. Users can thereby take advantage of the context created by an information search to editing without having to switch to the KANDOR level. They first use retrieval by reformulation to find an item that is similar to the new one, copy it, and use it as a template. In this way, they know which categories and attributes are reasonable to use; and because they see examples, they better understand what a category or attribute means. Values can often be reused, or they can be selected from a list of alternative values. They can also be typed in on the keyboard, a feature providing the same support (automatic completion, etc.) as it does in the formulation of a query. The query specification itself, which contains concepts used in the description of the information store, can also be transformed into a new item.

Evaluation—HELGON has been a joint research effort combining innovative system design and

The screenshot shows the HELGON interface with several panels:

- Category Hierarchy:** A tree diagram starting with 'THING' (count 7). It branches into 'COMMAND' (8), 'INPUT FUNCTION' (7), 'MAIL', 'PERSON' (11), and 'LITERATURE' (6). 'LITERATURE' further branches into 'ARTICLE', 'BOOK', 'HARBOOK', 'INPROCEEDINGS', 'PROCEEDINGS', and 'TECHREPORT'.
- Bookmarks of Items:** A list of previously viewed items: FischerLemkeHeper-Lemke1988, FabianLemke1984, Ambrascet1.1986, AramsonCarroll1986, AAI-83, and A. Archbold.
- Query:** THING, LITERATURE, TECHREPORT, INSTITUTION: not IBM, INSTITUTION: Department of Comput, YEAR: 1988.
- Matching Items:** FischerLemkeHeper-Lemke1988, Schwartz1988.
- Example of the Matching Item:** THING, LITERATURE, TECHREPORT, FischerLemkeHeper-Lemke1988, AUTHOR: G. Fischer, A.C. Lemke, H. Nieper-Lemke, TITLE: Enhancing Incremental Learning Processes with Knowledge-Based Systems, INSTITUTION: Department of Computer Science, University of Colorado, ADDRESS: Boulder, CO, NUMBER: CU-CS-392-88, MONTH: March, YEAR: 1988.
- Control Panel:** Retrieve, Show Item, Require Category, Prohibit Category, Require Attribute, Prohibit Attribute, Load Database, Save Database.
- Command Log:** Helgon command: Require Attribute "YEAR" "1988", Helgon command: Retrieve, Helgon command: [empty]

Fig. 5. Literature references in HELGON.

cognitive theory. An empirical evaluation of the information retrieval component of an earlier version of HELGON showed (for details see [35]) that:

- Subjects wanted to directly enter information they knew from the beginning because finding the right example to be criticized can be tedious. This capability was therefore incorporated in a later version of HELGON.
- Subjects had problems understanding the terms used in the information store.
- Subjects had problems with the hierarchical organization of the information store.

These findings suggest extensions for future versions of HELGON. An explanation component describing the terms used can be added to increase users' understanding of the system model. We are also experimenting with an *inductive* retrieval algorithm, which returns a set of items that match the query to varying degrees [36]. Including spreading activation in the system implies that a larger portion of the task is shifted to the system.

5.2. SYSTEMS' ASSISTANT: supporting mixed initiative dialogs

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 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 questions 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, the typical knowledge-based system that forces a particular format of discussion upon the user is not. In this section, we will describe several versions of a knowledge-based system which assist users in rebooting a computer [42]. The initial version, called REBOOTER was a MYCIN-style expert system which turned out to be behaviorally unacceptable and led to the development of SYSTEMS' ASSISTANT.

5.2.1. REBOOTER. 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. A sample session with REBOOTER is described in Fig. 6.

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. The rules here help users determine what causes the 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 2–6 months of on-the-job training are done by our novice systems administrators before they are confident enough to reboot machines on their own. 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.

The knowledge base—The knowledge base of the REBOOTER, which excludes the user-interface, consists of a set of OPS5 production rules [37]. The interference 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, and 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 dialog session that runs on traditional CRT terminals. REBOOTER presents a series of questions that lead the user

```

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.

```

Fig. 6. A partial session with REBOOTER.

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 shown in Fig. 6.

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 dialog 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 [38] program was introduced into the medical establishment. When experts in a field use a knowledge-based system they need to feel that they have an active role in the process of deciding what actions should be taken. In REBOOTER, the dialog 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.

5.2.2. THE SYSTEMS' ASSISTANT: incorporating information volunteering. Our solution to this prob-

lem 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.

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 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? OF 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.

The system's knowledge is explicitly represented in a world model (see Fig. 7) with which the user interacts in a direct manipulation style [19].

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.

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.

Experiences with the SYSTEMS' ASSISTANT have demonstrated that the system provides the right kind of mixture between highly structured dialogs (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 system acceptable to the expert.

5.3. JANUS: design environments

Design environments provide a framework in which the domain experts can themselves articulate their relevant knowledge precisely enough that it can then be used by the computer not only to aid in problem solving in the domain but also made subject to peer review and revision.

JANUS [46, 39] allows designers to construct artifacts in the domain of architectural design and at the same time informs them about principles of design and the reasoning underlying them. This process, which we call *Informed Design*, integrates two design activities: construction and argumentation. Construction is supported by a knowledge-based graphical design environment (see Fig. 8) and argumentation is supported by a hypertext system (see Fig. 9).

JANUS provides a set of domain-specific building blocks and has knowledge about how to combine them into useful designs. With this knowledge it "looks over the shoulder" of users carrying out a specific design. If it discovers a shortcoming in users' designs, it offers criticism, suggestions and explanations and assists users in improving their designs through cooperative problem solving. JANUS is not an expert system that dominates the design process by generating new designs from high-level goals or resolving design conflicts automatically. Users control the behavior of the system at all times (e.g. the critiquing can be "turned on and off"), and if users disagree with JANUS, they can modify its knowledge base. An objective of JANUS is to blend the designer and the computer into a problem-solving team to produce cooperatively better designs than each of them can by working alone.

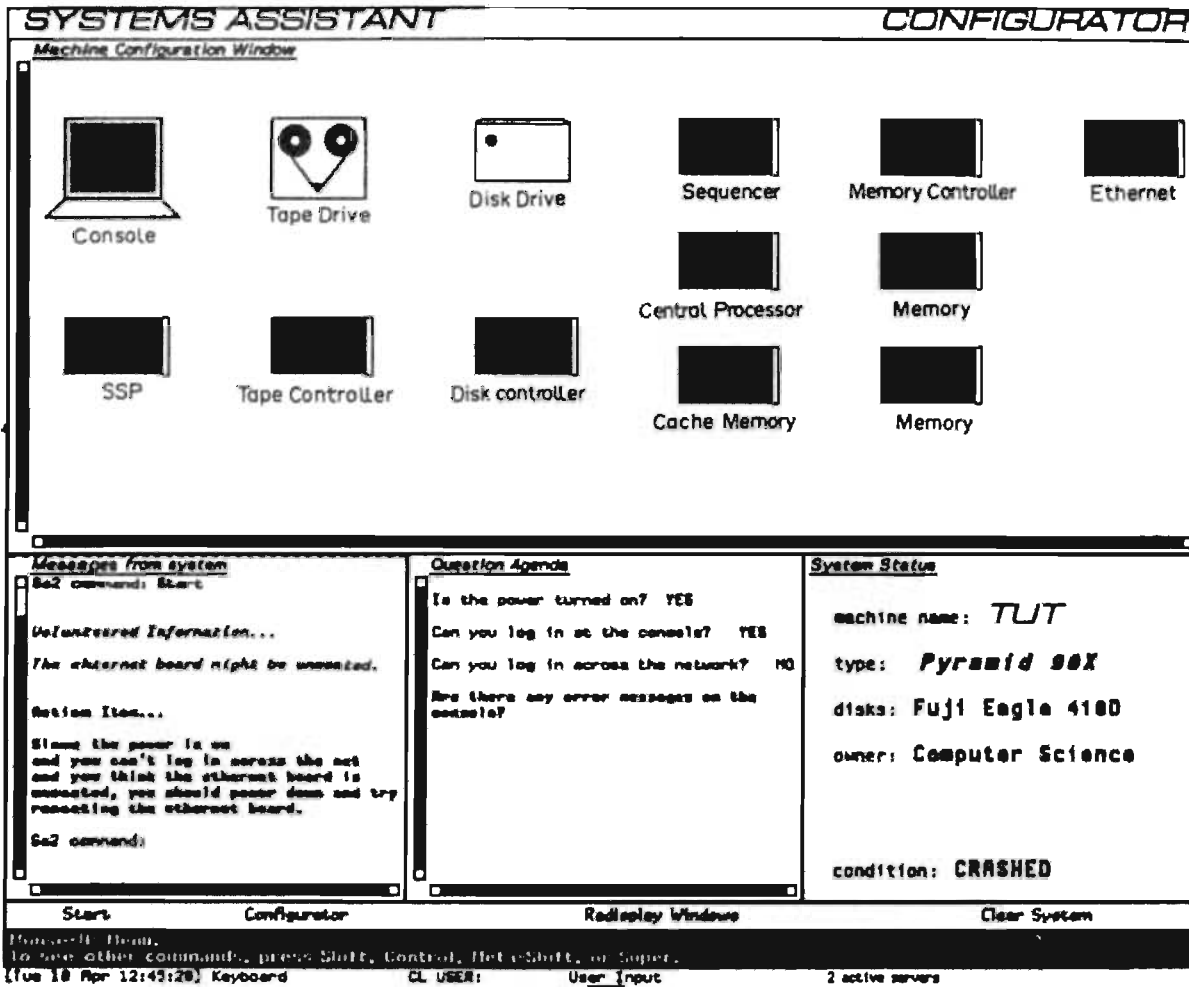


Fig. 7. The SYSTEM'S ASSISTANT in operation.

Critics in JANUS are procedures for detecting non-satisfying partial designs. JANUS' concept for integrating the constructive and argumentative component originated from the observation that the criticism generated by the critics is a limited type of argumentation. In particular, the construction actions can be seen as attempts to resolve design issues. For example, when a designer is positioning the sink in the kitchen, the issue being resolved is "Where should the sink be located"?

The knowledge-based critiquing mechanism in JANUS bridges the gap between construction and argumentation. This means means that critiquing and argumentation can be coupled by using JANUS' critics to provide the designer with immediate entry into the exact place in the hypertext network where the argumentation relevant to the current construction task lay. Such a combined system provides argumentative information for construction effectively, efficiently and without designers' having to realize they need information, suspect that needed information is in the system or know how to retrieve it.

JANUS' construction component—The constructive part of JANUS supports the construction of an artifact either "from scratch" or by modifying an already constructed artifact. To construct from scratch, the designer chooses building blocks from a design units "Palette" and positions them in the "Work area" (see Fig. 8).

To construct by modifying an existing artifact, the designer uses the "CATALOG" (lower left in Fig. 8), which contains many previously designed kitchens. The designer can browse through this catalog of examples until an interesting design is found. This design can then be selected and brought into the "Work Area", where it can be modified to the designer's liking.

The CATALOG contains two types of examples: "good" designs and "bad" designs. The former satisfy all the rules of kitchen design and will receive no negative criticism from the system's critics. People who want to design without knowing the underlying principles might want to select one of these, since minor modifications of these will be

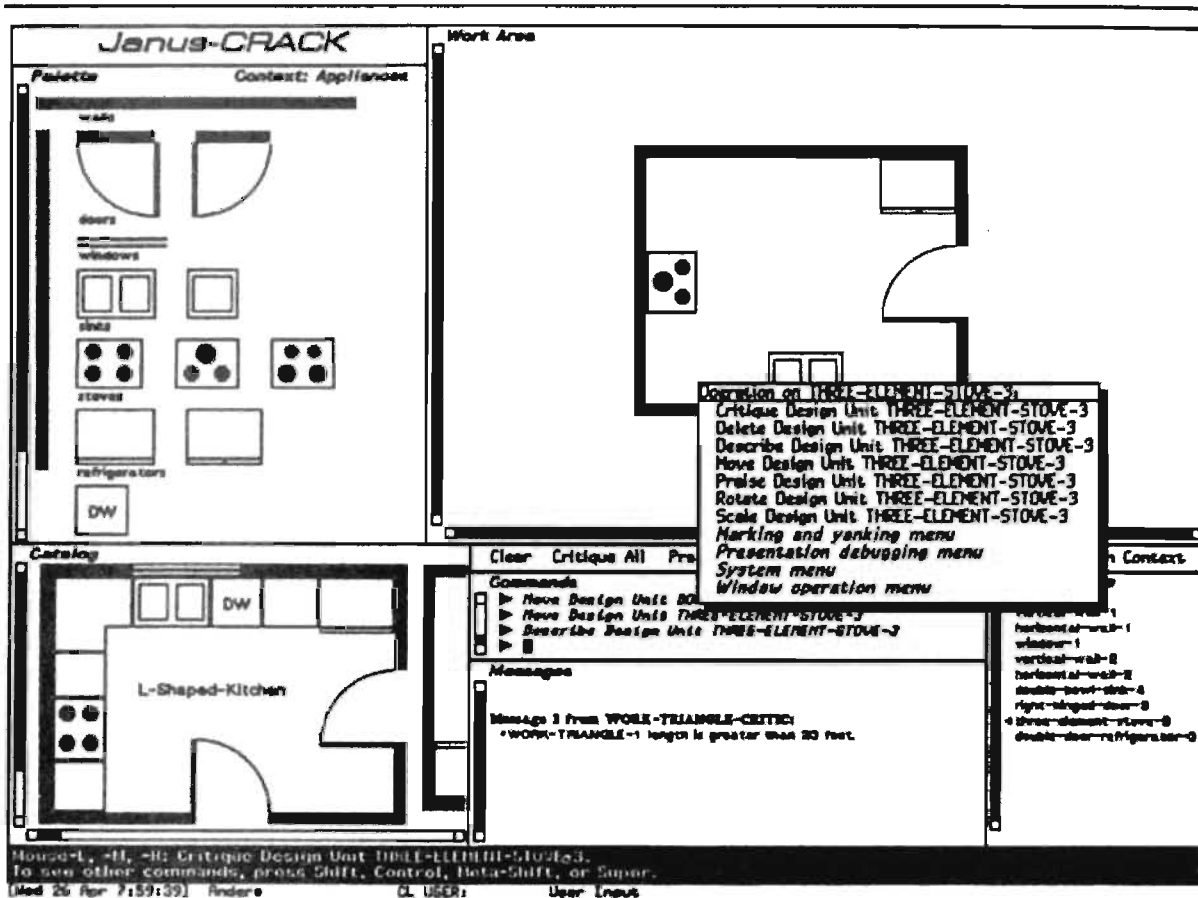


Fig. 8. JANUS construction interface.

probably result in little or no negative criticism from the critics.

The "bad" designs in the CATALOG are there to support learning about principles of design—in this case principles of kitchen design. By bringing these into the "Work Area", users can subject them to the critiquing by the system illustrating to them which principles of kitchen design are incorporated into the system.

The "good" designs in the CATALOG can also be used to learn design principles and explore their argumentative background. This can be done by bringing them into the "Work Area" then using the "Praise all" command. This command causes the system to display all of the rules which the selected example satisfies. This positive criticism also provides entry points into the hypertext argumentation.

JANUS' argumentation component—The hypertext component of JANUS is implemented using Symbolics' Concordia and Document Examiner software. Concordia is a hypertext editor [40] with which the issue base is implemented. The Document Examiner [41] provides functionality for on-line presentation and browsing of the issue base by users.

When users enter the argumentative part of JANUS, they are brought into a section of the issue base

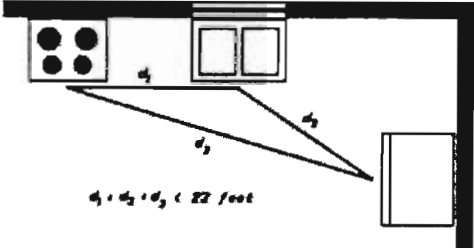
determined by and relevant to their current construction situation. They do not have to hunt for relevant information. Their point of entry into the hypertext network will contain relevant information. But since the argumentation on an issue can be large and complex, they can use the initial display of relevant information as the starting place for a navigational journey through the issue base. Following links will take them to additional relevant information. Upon completion of the examination of the relevant argumentative information the designer can return to construction and complete the current task.

Critics as hypertext agents—JANUS' knowledge-based critics serve as the mechanism linking construction to argumentation. They "watch over the shoulders" of designers while they are constructing and critique their work, displaying their criticism in the "Messages" pane (center bottom in Fig. 8) if design principles are violated. In doing so they also identify the argumentative context which is appropriate to the current construction situation.

For example, when a designer had designed the kitchen shown in Fig. 8, the "Work-Triangle-Critic" fires and detects that the work triangle is too large. To see the argumentation surrounding this issue, the

Janus-ViewPoints


Description (Work Triangle)
 The work triangle is an important concept in kitchen design. The work triangle denotes the center front distance between the three appliances: sink, stove and refrigerator. This length should be less than 23 feet to avoid unnecessary walking and to ensure an efficient work flow in the kitchen.



$d_1 + d_2 + d_3 < 23 \text{ feet}$

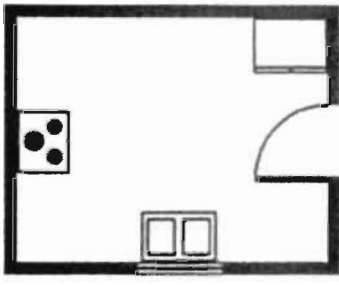
Figure 10: the work triangle

Answer (Refrigerator, Sink)
 The refrigerator should be near a sink, but not next to the sink.



Viewer: Default Viewer

Construction Situation



Visited Nodes

- Answer (Stove, Sink) Section
- Issue (Stove) Section
- Answer (Sink, Window) Section
- Description (Work Triangle) Section
- Answer (Refrigerator, Sink) Section

Commands

- ▶ Show Argumentation Description (Work Triangle)
- ▶ Show Outline Subissues (Equipment Area)
- ▶ Show Construction
- ▶

Show Outline Done
 Search For Topics Show Example
 Show Argumentation Show Counter Example
 Show Context Show Linking

© 1991, All show the current design under construction.
 To see other commands, press Shift, Control, Meta-Shift, or Super.
 (Used 26 Apr 1994) Anders GL UG&H User Issues

Fig. 9. JANUS argumentation interface.

designer has only to click on the text of this criticism with the mouse. The argumentative context shown in Fig. 9 is then displayed.

Evaluation of JANUS—We have evaluated JANUS with subjects ranging from neophyte to expert kitchen designers and from neophyte to expert computer users. We found that no user group had any significant overall advantage in using the system. Design students were more familiar with the general application domain but learned to use the system without much difficulty after some initial practice. Computer science students were able to understand the critics and learn from them to create reasonable kitchen designs. Users uncertain about criticism from the system or interested in more background information about design principles entered the hypertext system.

6. CONCLUSION

Wiener [17] concludes about the future of cockpit automation:

“The rapid pace of introduction of computer-based devices into the cockpit has outstripped the ability of designers, pilots and operators to formulate an overall strategy for their use and implementation.

The human factors profession is struggling to catch up. The devices themselves are highly reliable, but therein may lie the problem: they are also dumb and dutiful. This property of digital devices, and the fallibility of the human operator, has created a problem at the human-device interface. Putting ‘just one more computer’ into the cockpit is not the answer. The solution will come from a long, expensive, and sometimes tedious effort to develop a harmonious crew-automation interface, guided by an overall design philosophy.”

To take advantage of the potential of the great computing power available to us, we must develop joint human-computer systems as cooperative problem solving systems taking the goals of the human, the strengths and the weaknesses of humans and computers and the nature and structure of the task into account.

Acknowledgements—The author would like to thank his colleagues and students at the University of Colorado, Boulder, especially Helga Neiper-Lemke, who designed and developed HELGON, Curt Stevens, who designed and developed SYSTEMS’ ASSISTANT, Anders Morch and Raymond McCall, who designed and developed JANUS and Brent Reeves who carried out the “McGuckin” Study.

The research was partially supported by Grant No. DCR-8420944 from the National Science Foundation, Grant No. MDA903-C0143 from the Army Research Institute and grants from the Intelligent Systems Group at NYNEX and from Software Research Associates (SRA), Tokyo.

REFERENCES

- [1] M. J. Stefik. The next knowledge medium. *AI Mag.* 7(1), 34–36 (1986).
- [2] G. Fischer. Cooperative problem solving systems. *Proc. Ist Simp. Int. de Inteligencia Artificial*, Monterrey, Mexico, October, pp. 127–132 (1988).
- [3] T. W. Malone, K. R. Grant, K.-Y. Lai, R. Rao and D. Rosenblitt. Semi-structured messages are surprisingly useful for computer-supported coordination. *Proc. Conf. on Computer-Supported Cooperative Work (CSCW'86)*, MCC, Austin, Texas, pp. 102–114 (1986).
- [4] B. L. Webber and T. W. Finin. In response: next steps in natural language interaction. *Artificial Intelligence Applications for Business* (W. Reitman, Ed.), pp. 211–234, Chap. 12. Norwood, New Jersey (1984).
- [5] I. Greif (Ed.). *Computer-Supported Cooperative Work: A Book of Readings*. Morgan-Kaufmann, San Mateo, Calif. (1988).
- [6] A. H. Bond and L. Gasser (Eds). *Readings in Distributed Artificial Intelligence*. Morgan-Kaufmann, San Mateo, Calif. (1988).
- [7] D. D. Woods. Cognitive technologies: the design of joint human-machine cognitive systems. *AI Mag.* 6(4), 86–92 (1986).
- [8] G. Fischer. Cognitive view of reuse and redesign. *IEEE Software* 4(4), 60–72 (1987).
- [9] D. McCracken. Man + computer: a new symbiosis. *Commun. ACM* 22(11), 587–588 (1979).
- [10] M. Kay. The proper place of men and machines in language translation. Technical Report CSL-80-11, Xerox Palo Alto Research Center (1980).
- [11] A. B. Chambers and D. C. Nagel. Pilots of the future: human or computer? *Commun. ACM* 28(11), 1187–119 (1985).
- [12] R. G. Smith. On the development of commercial expert systems. *AI Mag.* 5(3), 61–73 (1984).
- [13] M. Stelzner and M. D. Williams. The evolution of interface requirements for expert systems. *Expert Systems: The User Interface* (J. A. Hendlar, Ed.), pp. 285–306, Chap. 12. Ablex, Norwood, New Jersey (1988).
- [14] J. R. Carbonell. AI in CAI: an artificial-intelligence approach to computer-assisted instruction. *IEEE Trans Man-Machine Systems*. MMS-11(4), (1970).
- [15] G. Fischer, A. C. Lemke and T. Schwab. Knowledge-based help systems. *Human Factors in Computing Systems, CHI'85 Conf. Proc.*, San Francisco, pp. 161–167. ACM, New York (1985).
- [16] L. A. Suchman. *Plans and Situated Actions*. Cambridge University Press, New York (1987).
- [17] E. L. Wiener. Cockpit automation. *Human Factors in Aviation* (F. L. Wiener and D. C. Nagal, Eds), pp. 433–461. Academic Press, San Diego (1988).
- [18] G. Fischer and A. C. Lemke. Construction kits and design environments: steps toward human problem-domain communication. *Human-Computer Interaction* 3(3), 179–222 (1988).
- [19] E. L. Hutchins, J. D. Hollan and D. A. Norman. Direct manipulation interfaces. *User Centered System Design, New Perspectives on Human-Computer Interaction* (D. A. Norman and S. W. Draper, Eds), pp. 87–124, Chap. 5. Lawrence Erlbaum, Hillsdale (1986).
- [20] T. Winograd and F. Flores. *Understanding Computers and Cognition: A New Foundation for Design*. Ablex, Norwood, New Jersey (1986).
- [21] M. J. Bates and R. J. Bobrow. Natural language interfaces: what's here, what's coming and who needs it. *Artificial Intelligence Applications for Business* (W. Reitman, Ed.), pp. 179–194, Chap. 10. Ablex, Norwood, New Jersey (1984).
- [22] G. Robertson, D. McCracken and A. Newell. The ZOG approach to man-machine communication. *Int. J. Man-Machine Studies* 14, 461–488 (1981).
- [23] G. Fischer and T. Mastaglio. Computer-based critics. *Proc. Twenty-Second Annual Hawaii Conf. on System Sciences, Vol. III: Decision Support and Knowledge Based Systems Track*, pp. 427–436. IEEE Computer Society (1989).
- [24] B. Reeves. Finding and choosing the right object in a large hardware store—an empirical study of cooperative problem solving among humans. Technical Report, Department of Computer Science, University of Colorado, Boulder (1989).
- [25] J. R. Carbonell. Mixed-initiative man-computer instructional dialogues. Report 1971, BBN (1970).
- [26] J. R. Anderson and B. J. Reiser. The LISP tutor. *BYTE* 10(4), 159–175 (1985).
- [27] J. M. Carroll and J. McKendree. Interface design issues for advice-giving expert systems. *Commun. ACM* 30(1), 14–31 (1987).
- [28] G. Fischer. A critic for LISP. *Proc. 10th Int. Joint Conf. on Artificial Intelligence*, Milan, Italy (J. McDermott, Ed.), pp. 177–184. Morgan-Kaufmann, Los Altos, Calif. (1987).
- [29] G. Fischer and A. Morch. CRACK: a critiquing approach to cooperative kitchen design. *Proc. Int. Conf. on Intelligent Tutoring Systems*, Montreal, Canada, pp. 176–185. ACM, New York (1988).
- [30] R. A. Bolt. *The Human Interface*. Lifetime Learning Publications, Belmont, Calif. (1984).
- [31] G. Fischer, P. W. Foltz, W. Kintsch, H. Nieper-Lemke and C. Stevens. Personal information systems and models of human memory. Technical Report, Dept of Computer Science, University of Colorado (1989).
- [32] G. Fischer and H. Nieper-Lemke. HELGON: extending the retrieval by reformation paradigm. *Human Factors in Computing Systems, CHI'89 Conf. Proc.* Austin, Texas, pp. 357–362. ACM, New York (1989).
- [33] M. D. Williams. What makes RABBIT Run? *Int. J. Man-Machine Studies* 21, 333–352 (1984).
- [34] F. G. Halasz. Reflections on notecards: seven issues for the next generation of hypermedia systems. *Commun. ACM* 31(7), 836–852 (1988).
- [35] P. W. Foltz and W. Kintsch. An empirical study of retrieval by reformulation on HELGON. *Mental Models and User-Centered Design* (A. A. Turner, Ed.). Workshop Report (Breckenridge, CO), Institute of Cognitive Science, University of Colorado, Boulder, Technical Report No. 88–9, pp. 9–14 (1988).
- [36] M. C. Mozer. Inductive information retrieval using parallel distributed computation. ICS Report 8406, Institute for Cognitive Science, University of California, San Diego (1984).
- [37] L. Brownston, R. Farrel, E. Kant and N. Martin. *Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming*. Addison-Wesley, Reading, Mass. (1985).
- [38] B. G. Buchanan and E. H. Shortliffe. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, Reading, Mass. (1984).
- [39] G. Fischer, R. McCall and A. Morch. Design environments for constructive and argumentative design. *Human Factors in Computing Systems, CHI'89 Conf. Proc.*, Austin, Texas, pp. 269–275. ACM, New York (1989).
- [40] J. H. Walker. Supporting document development with Concordia. *IEEE Comput.* 21(1), 48–59 (1988).

- [41] J. H. Walker. Document examiner: delivery interface for hypertext documents. *Hypertext'87 Papers*, University of North Carolina, Chapel Hill, pp. 307-323 (1987).
- [42] G. Fischer and C. Stevens. Volunteering information—enhancing the communication capabilities of knowledge-based systems. *Proc. INTERACT'87, 2nd IFIP Conf. on Human-Computer Interaction* (Stuttgart, F.R.G.) (H.-J. Bullinger and B. Shackel, Eds), pp. 965-971. North-Holland, Amsterdam (1987).
- [43] G. Fischer, A. C. Lemke, T. Mastaglio and A. Morch. Using critics to empower users. *Human Factors in Computing Systems, CHI'90 Conf. Proc.*, Seattle, WA, ACM, New York (1990).
- [44] G. Fischer and C. Stevens. Information access in complex, poorly structured information spaces. Technical Report, Department of Computer Science, University of Colorado (1990).
- [45] S. Mannes and W. Kintsch. Planning routine computing tasks: understanding what to do. Technical Report 89-8, Institute of Cognitive Science, University of Colorado (1989).
- [46] G. Fischer, R. McCall and A. Morch. JANUS: integrating hypertext with a knowledge-based design environment. *Proc. Hypertext'89*, ACM, New York, pp. 105-117 (1989).