HUMAN-COMPUTER INTERACTION, 1992, Volume 7, pp. 281-314

Supporting Indirect Collaborative Design With Integrated Knowledge-Based Design Environments

Gerhard Fischer
University of Colorado

Jonathan Grudin
University of California, Irvine

Andreas Lemke
GMD, Darmstadt, Germany

Raymond McCall, Jonathan Ostwald, Brent Reeves, and **Frank Shipman**
University of Colorado

ABSTRACT

We are developing a conceptual framework and a demonstration system for collaboration among members of design teams when direct communication among these members is impossible or impractical. Our research focuses on the long-term, indirect communication needs of project teams rather than the

Authors' present addresses: Gerhard Fischer, Raymond McCall, Jonathan Ostwald, Brent Reeves, and Frank Shipman, Department of Computer Science and Institute of Cognitive Science, University of Colorado, Boulder, CO 80309; Jonathan Grudin, Information and Computer Science Department, University of California, Irvine, CA 92717; Andreas Lemke, GMD-F4, Dolivostrasse 15, D-6100 Darmstadt, Germany.

CONTENTS

short-term needs of face-to-face communication or electronic mail. We address these needs with integrated, domain-oriented design environments. Our conceptual framework and our system-building efforts address two major issues: (a) How does individual work blend into project work (especially in large projects that span great distances and time)? and (b) What role do the work objects play in this coordination? We use a specific domain-oriented design environment (NETWORK-HYDRA—for the design of computer networks) to illustrate our approach, and we discuss HYDRA as the underlying domain-independent, multifaceted architecture for design environments.

1. INTRODUCTION

Support for long-term collaboration is important in modern technologically oriented design projects. Such projects are increasingly large, complex, and long in duration. System design itself can extend over many years, only to be followed by extended periods of maintenance and redesign. Specialists from many different domains must coordinate their efforts despite large separations of distance and time. In such projects, constructive collaboration is crucial for success but is increasingly difficult to achieve. This difficulty is due

in large part to ignorance by individual designers of how the decisions they make interact with decisions made by other designers. Much of this, in turn, consists of simply not knowing what has been decided and why. Our system will enable designers to be informed about such things within the context of their work on real-world design problems. With our systems, coordination does not take place in a separate phase and place (e.g., in meetings) but is integrated into designers' individual work in their workplace. The systems we develop let users construct solutions to design problems, advise them when they are getting into trouble, and then provide directly relevant information for understanding how to cope with such trouble.

We are developing a domain-oriented prototype design environment, NETWORK-HYDRA (or NETWORK for brevity), that supports the design of computer and communication networks and instantiates our multifaceted, domain-independent architecture, HYDRA. The network domain is useful for our purposes because it requires interdisciplinary project teams and deals with artifacts that are repeatedly redesigned over years of use.

Our approach is based on experience with a number of previous prototypes we have built, together with an analysis of previous efforts by others attempting to support collaborative work. Our approach differs from the latter in three major respects: (a) it integrates collaboration into individual work; (b) it covers not only the argumentative aspect of design but integrates construction, reuse, and specification; and (c) it supports the creation of design rationale by providing preexisting issue-based hypermedia systems of domain-specific argumentation that users can modify.

In this article, we first describe problems in supporting long-term coordination and collaboration. We then discuss a conceptual framework for integrated, domain-oriented design environments that addresses these problems. A scenario illustrates the use of the system. The individual components and the links of HYDRA are then described. We conclude by discussing issues that will be explored in our future research efforts.

2. PROBLEMS IN FACILITATING LONG-TERM COORDINATION AND COLLABORATION AMONG DESIGNERS

A large percentage of problem-solving tasks can be categorized as design tasks. These include the design of buildings, bridges, and electromechanical devices; the development of computer hardware and software; and the creation of application systems and operating environments using computer technology. Designing artifacts well is a difficult task, especially due to the explosive growth of technology and the resultant increase in complexity of design problems. Even experts rely on the expertise of others in the process of design by referring to textbooks, standards, legal constraints, and especially

previous design efforts. Project complexity forces large and heterogeneous groups to work together on projects over long periods of time.

In technologically oriented design fields, the knowledge base needed for design grows and changes at an alarming rate (Draper, 1984; Fischer, 1988). It includes not only knowledge about the design process but also knowledge about artifacts of that process—parts used in designing artifacts, subassemblies previously created by other design efforts, and rationales for previous design decisions.

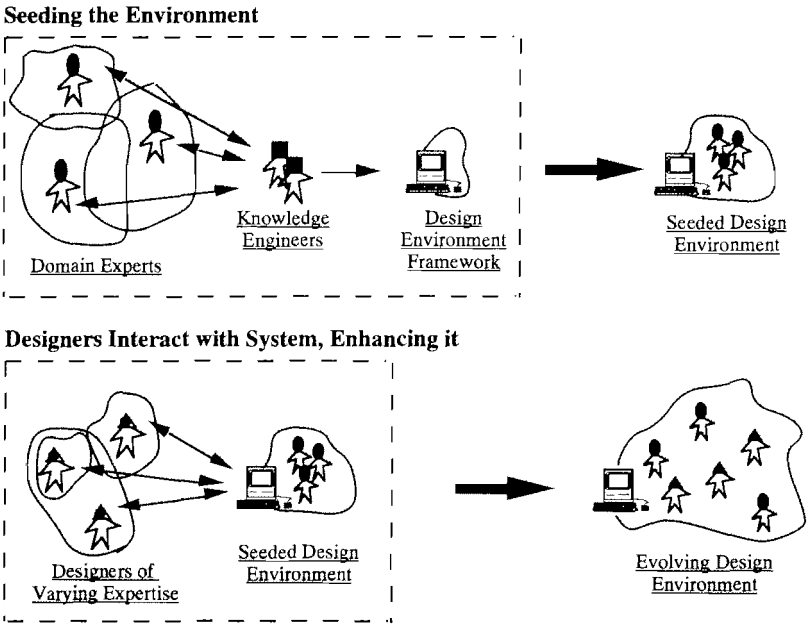
Given this situation, designers generally have a limited awareness and understanding of how the work of other designers within the project (or in similar projects) is relevant to their own part of the design task. The large and growing discrepancy between the amount of such relevant knowledge and the amount any one designer can know and remember puts a limit on progress in design. Overcoming this limit is a central challenge for developers of systems that support collaborative design.

A major portion of the work of project teams is done by team members working individually rather than in groups. Most design problems are at best nearly decomposable (Simon, 1981). For example, computer network design is dependent on the physical structure of buildings, but even though these interactions are weaker than the interactions among different parts of a network, they cannot be ignored. Team members need to coordinate the work they do as individuals. For example, it is crucial that conflicts between individuals be detected before much work has been based on the conflicting decisions. Beyond the mere elimination of conflicts, coordination should extend to positive collaboration. Meetings and other types of direct communication are among the most used means for coordination and collaboration, but in many situations (especially ones involving long-term collaboration) these are not feasible. Modern design projects can extend over many years and can involve a high turnover in personnel. People who are not in the project group at the same time need to coordinate and collaborate in the design of a system. Much of the design work on systems is done as maintenance and redesign, and the people doing this work are often not members of the original design team. To be able to do this work well or at all, however, requires collaboration with the original designers of the system.

Even when designers work in parallel, the time frame for individual decision making often does not encourage communication, either face to face or by electronic mail. When it does allow it in principle, the resulting disruption (resulting from being forced to leave the context of the work) to individual work sometimes militates against it.

Support is needed for indirect coordination and collaboration beyond what electronic mail and most proposed *computer-supported cooperative work* (CSCW) software could provide, even in principle. This support should allow team members to work separately (across substantial distances in space and time)

Figure 1. **Indirect communication in a knowledge-based design environment. In the first stage, a seed for the knowledge-based design environment is constructed. This is done by domain experts who carry out a participatory design process with knowledge engineers (e.g., in our project, we interact with domain experts in network design). In later stages, the environment (serving as a design memory, an institutional memory, and a design history) will grow through interaction with designers of varying expertise. The growth will occur by a dialectic process between the information contained in the design environment and the new tasks and demands.**



but alert them to the existence of potential interactions between their work and the work of others. Where such interactions exist, support should be provided for collaboration and conflict resolution. Designers must be able to interact with representations of design rationale created by previous designers (see Figure 1). Technology enabling this could effectively create virtual cooperation between all designers who ever worked on the project.

2.1. Indirect Communication

Long-term indirect communication for coordination of individual work in groups can be achieved by using a *group memory*—a collection of shared information repositories containing a cumulative record of rationale, solution components, and information about prior projects. There are two crucial

issues concerning such a memory. One is how information gets into the memory and accumulates, and the other is how information in the memory is made available to the individual designer. Both of these issues present formidable obstacles for development of technology for long-term collaboration.

Perhaps the single most difficult problem in getting information into the various components of group memory is that of motivating designers to impart this information. Nowhere is this problem more difficult than in the input of design rationale. The designer must regard the time and effort invested in contributing information to be immediately worthwhile for the task at hand—not merely for someone else or for some putative long-term gain (Grudin, 1988). To minimize the amount of information that designers themselves must add, design environments must be *seeded* (see Figure 1).

An important principle for our approach is that designers are more likely to use and to add to group memory of design rationale if they do not have to create project rationale entirely from scratch. Seeding the base of project rationale by putting in substantial amounts of domain-oriented rationale in advance reduces the designer's required input to that rationale specific to the project at hand. In addition, it rewards designers with useful information on aspects of the domain in which they do not have expertise.

The seed must contain information of sufficient amount and quality to elicit rationale from designers in the form of commentary and responses to issues. This does not mean, however, that the seed needs to be complete or correct in all aspects. Our experience in building and using issue bases of design rationale over the past decade (McCall, 1991) suggests that limited inadequacies in these respects provoke corrective input from the designers. The open collection of information contained in group memory can be strongly contrasted with the knowledge base in an expert system, whose completeness and correctness are crucial for its usefulness.

2.2. Relating Group Memory to Individual Work

The issue of how to make information in group memory available to designers presents formidable challenges. It is not possible for those who work on complex projects to know the entire group memory (i.e., all the decisions and rationales for a project). Designers need to know only about those things that are relevant to their own work (e.g., potential conflicts between their work and the work of others). Computer systems that support individual work must actively alert designers to the existence of relevant information (Fischer & Nakakoji, 1991). This leaves designers free to concentrate on their tasks and lets them deal with problems of coordination only when the need to do so arises.

Merely being able to present such information when appropriate is not enough. Viewing design as situated action (Schön, 1983; Suchman, 1987; Winograd & Flores, 1986) implies that the prestored collections for design rationale, principles, precedents, and other information in group memory cannot account for all information that designers must take into account. Designers must evaluate and interpret the contents of group memory in light of the current design task situation, adding to and modifying those contents where appropriate. The ideas of all theorists of situated cognition are incompatible with the notion of delegating design judgment to the computer (e.g., using expert systems)—a prevalent notion in the literature on computer-supported design (Gero, 1990). Their theories instead require that computer-supported design systems leave room for human judgment at all times (Rittel, 1984).

Integrating Action and Reflection. The design methodologists mentioned earlier all subscribe to an "action-breakdown-repair model" of the relationship between action and rational thought. Schön (1983) went the furthest in detailing this model and in describing its central role in design. He saw design as involving repeated alternation between situated action and the type of reflection he called *reflection in action.* Design is a nonreflective process and continues until there is a *breakdown.* This happens when nonreflective action results in unanticipated consequences—either good or bad. Reflection is used to repair the breakdown, and then nonreflective action continues. Reflection in action takes place within the *action present,* that is, the time period when the decision to act has been made but the final decision about how to act has not. This is the time period during which reflection can still affect what action is taken. This contextualization distinguishes reflection in action from the reflection of preplanning and postmortem analysis.

The model of design as reflection in action provides a basis for determining how group memory should be related to individual work. A breakdown occurs when individual work conflicts with the contents of group memory. The situation must "talk back" (Schön, 1983, p. 79) to the individual designer in order for the interaction with group memory to be understood, dealt with, and repaired within the action present. The repair process may trigger new insights or reaction from the designer, which can be added to group memory and thus be made available to the group as a whole.

Integrating Construction and Argumentation. Our work starts with the principles mentioned in the previous sections and asks, "How can these principles be facilitated in computer-based tools?" Schön's ideas do not in themselves tell us what the architecture of a system for indirect design collaboration should be. His concepts must be further operationalized and augmented to provide a basis for computer-based systems. His interest is not

in building systems that assist designers in design tasks; he is interested in finding psychological explanations of the designer. His theory is descriptive, and it identifies the importance of human resources in this process.

Our interest is in understanding (a) how designers design; (b) how designers might organize designing so that they are more effective, avoid problems, and learn new things as they go along; and (c) how all of this can be supported by computational media (Fischer & Nakakoji, 1992). We have engaged in something that Schön's theory considers important—building objects to think with in the form of demonstration prototypes (Fischer, McCall, & Morch, 1989b; Lemke & Fischer, 1990; McCall et al., 1990) to test the theory in practice, to experience breakdowns of the theory, and, as a consequence, to refine the theory when necessary. In our work, we have demonstrated that computational mechanisms can be created that can take some of Schön's concepts and bring them alive in a computational environment.

In our system-building efforts, we interpret action as *construction* and reflection as *argumentation.* This creates the problem of integrating construction and argumentation. Construction is the process of shaping the solution (i.e., manipulating form). Argumentation is the process of reasoning about the problem and its solution.

Along with support for construction and argumentation, users need support for perceiving breakdowns. Experiences with our early systems (Fischer, McCall, & Morch, 1989a) indicate that users often do not hear the situation talk-back (i.e., breakdowns are not always apparent to the user). This problem is compounded when individuals design in the context of a group project because they cannot have perfect knowledge of the entire project. To integrate construction and argumentation, the system must play an active role in alerting individual designers to breakdowns as well as providing argumentative support for coping with the breakdowns.

2.3. Related Work

The goal of our work is the exploration of indirect methods for collaboration and coordination within the working context (Ehn, 1988). Coordination can be mediated through explicit means, namely, messages sent at definite times to specified recipients. Alternatively, coordination can be achieved by creating *shared information spaces* containing versions of the artifact being constructed, argumentation structures, and design rationale. We have chosen shared information spaces so that coordination can occur without interrupting the designer's concentration on the design situation. Our emphasis on the coordination within the work environment and with the design activities centered around the evolving artifact is an important difference between our

project and previous research on supporting collaborative work (Conklin & Begeman, 1988).

Support systems in collaborative design must have functionality that includes but goes beyond that of traditional computer-aided design (CAD) systems, task coordination systems, and purely issue-based systems. In this section, we discuss the strengths and limitations of these classes of systems in supporting collaborative design as well as specific related work involving knowledge-based and hypermedia technology.

Traditional Computer-Aided Design Systems. CAD systems support the form-constructing activities of design. They do not, however, support coordination of the work of individual designers with the work of the rest of a project group. Although some CAD systems provide computer-based tools to help evaluate a design (Steele, 1987), most conventional CAD systems lack the ability to critique the solutions whose constructions they facilitate.

Expert Systems. R1/XCON demonstrated that support for computer configuration is possible using an expert system (McDermott, 1982). The differences between the configuration task and the design task are related to the greater freedom within the design task. Configuration can be characterized as a highly constrained form of design. In contrast, a designer often has many degrees of freedom and places constraints on the design that serve to focus the design. These constraints may later be strengthened or weakened by the designer as the design is progressing. It is the freedom of the designer to remove and add constraints that makes design a wicked problem (Rittel & Webber, 1984). Our design environments include knowledge-based critics similar to the rules found in an expert system. In our conceptual framework, critiquing is used to empower designers (e.g., by informing them of interactions with the group knowledge base) and not to replace them (Fischer, Lemke, Mastaglio, & Morch, 1991).

Synchronous Support Systems. Much research in supporting collaborative work has gone toward supporting synchronous communication, for example, COLAB (Stefik et al., 1987) and GROVE (Ellis, Gibbs, & Rein, 1991). Supporting indirect communication is equally important for good design. Direct communication is not possible when the designer responsible for a previous design decision is no longer with the project or the company. A domain in which extended project lifetimes would benefit from indirect communication is software design, in which empirical studies (Computer Science and Technology Board [CSTB], 1990) have shown that 40% to 60% of the development of complex software systems goes into maintenance. Our notion of the evolutionary development of a design environment (see Figure 1) is especially appropriate, because the National Institute of Standards and

Technology and the U.S. Air Force have found that up to 75% of the total maintenance effort is enhancement in complex systems (CSTB, 1990).

Systems for Structured Communication. The most successful asynchronous coordination tool to date is electronic mail. Extensions to the electronic mail paradigm include Winograd's work on the theory of conversation (Winograd, 1988) and Malone's work with semistructured messaging systems (Malone, Grant, Lai, Rao, & Rosenblitt, 1986, 1988). Adding structure and other methods of improvement to electronic mail can certainly help coordination to some degree. One undesirable effect of using these systems for collaboration in design is that the designer must interrupt the construction situation (i.e., leave the design tool). Our design environments inform the designer within the context of the construction situation, where new information can be put to use immediately. By having the communication mechanism as part of the design environment, the contents of the communication can include references to parts of the designed object and associated rationale and can provide a means for integrating the graphical representations, formal and semiformal domain knowledge, and design history into messages.

Another difficulty can arise if the message system is separated from the design environment. If designers need more information on another part of the project, they have to send a message and wait for a reply, rather than being able to consult directly with a group memory, as in our approach.

Text-Based Information Systems. The Arizona Analyst Information System (AAIS; Lynch, Snyder, Vogel, & McHenry, 1990) provides practical experience in the area of using group memories. AAIS is a text-based system that goes beyond linear text and printed documentation. It consists of several on-line information bases on specific topics built up over several years by groups of researchers, each item consisting of a roughly file-card-size text item that is accessible through a hierarchical index. The system is passive, providing no critiquing or design support of any kind, but it represents several years' experience with the collaborative creation, use, and maintenance of a large information base organized around issues. The largest information base has more than 40,000 entries from 20,000 sources and has served more than 50 researchers. The AAIS databases have supported 3 dissertations, 54 refereed papers, 19 book chapters, and reports that include policy analyses for the U.S. Congress.

Hypermedia Systems. Hypermedia (Halasz, 1988; McCall, 1989) is a promising technology for use in collaborative design environments. Hypermedia systems provide a technology well suited for storing group

memories beyond the text-based level in the AAIS. Unique characteristics of hypermedia are (a) the multiplicity of connections between media fragments as opposed to the linear structure of traditional text and (b) the availability of media other than text. The user modifiable connectivity of hypermedia documents provides quick access to explanatory, elaborative, and other related information. New media (e.g., graphics, animation, and sound) are more effective than text in conveying certain kinds of information such as two- and three-dimensional spatial relationships as well as processes, behaviors, and evolution of systems.

The DOCUMENT EXAMINER is a delivery interface for commercial hypertext documents (Walker, 1987) that handles the on-line documentation for the Genera Software Environment. As a delivery vehicle, the DOCUMENT EXAMINER was intended to be a reader's interface only. Documentation was to be prepared by product developers using a separate interface. By using the DOCUMENT EXAMINER as an argumentation component in previous system-building efforts (Fischer et al., 1989b), we have found that the asymmetry between the reader's and writer's interfaces limits the DOCUMENT EXAMINER's ability to support the evolution of a design environment seed. A design environment expected to evolve through the contributions of its users must have a single interface, allowing both reading and writing by the users.

Systems such as KMS (Akscyn, McCracken, & Yoder, 1988) and VNS (Shipman, Chaney, & Gorry, 1989) include an integrated reading and writing interface, which provides for the arbitrary organization of multimedia information. These systems are general purpose and are not integrated with design tools.

In contrast to the general purpose, unstructured approach of KMS and VNS, there are a number of systems for supporting the collection and use of design rationale. These systems provide a design rationale representation language, which consists of many node and link types with which to encode the design rationale. The gIBIS system (Conklin & Begeman, 1988) supports group communication in the form of issue-based design rationale through a graphical interface. SIBYL (Lee, 1990) includes a more formal representation language for design rationale than gIBIS or our system, and it emphasizes services like dependency, plausibility, viewpoint, and precedent management. In one observational study (Yakemovic & Conklin, 1990), the recording of design rationale was found to provide long-term rewards if the initial work of recording the design rationale could be accomplished. It is unclear how well the current generation of design rationale systems work in practice without strong support internal to a project for the approach. Our work addresses this issue by integrating the design rationale into the construction environment. This makes potential benefits more obvious to designers (Fischer, Lemke, McCall, & Morch, 1991).

Computer-Supported Cooperative Work. Most of the preceding categories are found under the broader categories of CSCW (Greif, 1988; Johansen, 1988). Most CSCW applications do not focus on shared information spaces, lack the integration of individual users' roles into the overall process, and do not represent the evolving artifact itself. These issues may be responsible for the limited success of CSCW applications (Ellis, 1991; Grudin, 1990; Sorgaard, 1988). For example, the constraints on the communication process imposed by the process model embodied in the COORDINATOR often lead to rejection of the application or to ignoring of advanced features (Bullen & Bennett, 1990; Erickson, 1989). The explanation for this may be found in the difficulty of providing computer support flexible enough to handle the variable and transitory nature of roles and processes in most group work situations (Grudin, in press). In general, these findings provide support for our focus on shared information systems.

A related distinction of our approach within the CSCW field is our focus on benefits for individual designers. A common source of failures for CSCW applications is that the primary intended beneficiaries are managers, whose use of systems is often less than individual contributors for whom they generate more work and less direct benefit (Grudin, 1988). It is critical to reduce the burden of use and to provide benefits for the principal users of the system. Several of the components of our system described later, notably the domain knowledge seed, must be entered into the system once, but subsequent reuse provides a substantial benefit for future design activities.

3. SUPPORTING COLLABORATIVE DESIGN WITH INTEGRATED KNOWLEDGE-BASED DESIGN ENVIRONMENTS

To further develop and test our approach for supporting indirect collaboration among group members situated in the work context, we are developing the specific domain-oriented design environment NETWORK, which supports the design of computer networks. NETWORK is an instantiation of HYDRA, the underlying domain-independent, multifaceted architecture for design environments. In this section, we briefly discuss the conceptual framework underlying the system and describe the chosen application domain.

3.1. Conceptual Framework

Our previous work on design environments, such as FRAMER (Lemke & Fischer, 1990), JANUS (Fischer et al., 1989b), and PHIDIAS (McCall et al., 1990), has emphasized systems for individual designers by providing support

of human problem-domain communication (Fischer & Lemke, 1988) and construction kits.

Design Environments. Human problem-domain communication and construction kits are necessary but not sufficient for good design. Upon evaluating prototypical construction kits (Fischer & Lemke, 1988), we found that they do not by themselves assist the user in constructing interesting and useful artifacts in the application domain. To do this, they need knowledge to distinguish "good" designs from "bad" designs. Design environments combine construction kits with *critics* (Fischer, Lemke, McCall, & Morch, 1991). Critics use knowledge of design principles for the detection of suboptimal solutions constructed by the designer. One of the challenges for critiquing systems is avoiding work disruption. Our systems accomplish this by making the critics sensitive to the specific design situation (Fischer & Nakakoji, 1991), incorporating a model of the user (Fischer, Lemke, McCall, & Morch, 1991), and giving users control over when and which critics should fire (Fischer & Girgensohn, 1990). JANUS has demonstrated that critics do not need to be highly intelligent (i.e., more intelligent than a designer) to be useful. Simple critics can be informative because they are based on domain knowledge that designers might not have (e.g., network designers are not necessarily familiar with relevant knowledge about fire codes for buildings).

Issue-Based Hypermedia. Design tasks are argumentative and open ended (Rittel, 1984); therefore, final designs addressing the same problem may take on many different forms. The large variety of solutions and questions that might be considered in a design task requires that large information spaces to be considered. In an attempt to improve design by improving the reasoning on which it is based, Rittel (1972) developed the IBIS (Issue-Based Information System) method (despite its name, IBIS is a method and not a software system) for documenting design rationale. With IBIS, information is organized around the discussion of questions, referred to as issues. The method of discussion is deliberation, that is, considering arguments for and against various proposed answers to the issues, these answers sometimes being known as positions.

We use a variant of the IBIS method, called PHI (Procedural Hierarchy of Issues; McCall, 1991), which extends Rittel's original IBIS by broadening the scope of the issue concept and by altering the structure of relating issues. IBIS uses a variety of relationships to connect issues. These include "more general than," "similar to," "logical successor of," "temporal successor of," and "replaces." PHI streamlines interissue structure by focusing on dependency relationships between issues, and it structures issues by the way in which a given issue is "served by" other issues, that is, the way in which the answer to the given issue depends on answers to other issues (Fischer, Lemke, McCall,

& Morch, 1991). The PHI method has been in nearly continual use by student designers and others for more than a decade. It is currently the basis for all student projects in a required course for undergraduates majoring in environmental design at the University of Colorado, Boulder. PHI has been applied to a wide range of projects, from health care policymaking to building and software design (McCall, Mistrik, & Schuler, 1981; McCall et al., 1990).

Domain Knowledge Seed. The initial collection of design parts, rules, and discussion included in the knowledge base is called the *seed.* The seed consists of domain knowledge in various forms (e.g., palette items, critics, argumentation, and a catalog of canonical designs). Our work addresses the question of what is the appropriate domain knowledge to be included in the seed. This question includes what objects should be in the construction palette, what examples should be in a catalog, what information about them is important to include, what critics should be initially in the design environment, and what issues and answers should be discussed in the issue-based hypermedia. Because the concept of a design environment seed is based on the concept of reusing design components and design rationale, our task is to identify reoccurring components and rationales in the network domain.

In the current version of NETWORK, we represent an existing network design (the campus network of the University of Colorado; see Figure 2). Our previous efforts (Fischer et al., 1989b) taught us that acquiring and representing domain knowledge without the guidance of real-life situations are tasks that are not focused enough and often turn out to be wasted efforts. By modeling an existing situation we are guided in our selection of domain knowledge that is relevant at least in one design situation (to determine the relevance of knowledge by potential use situations differentiates our effort from the CYC project; Lenat & Guha, 1990).

Design can be considered as a dialectical process between being in the tradition of a professional activity and at the same time transcending it (Ehn, 1988). Designing the seed is not merely the task of replicating existing design information. It is also the task of designing a system that will alter the way workers perform their job. If we are to alter the practice of network designers in a positive way, they must be partners in the design of the seed (see Figure 1). This requires the network domain experts to become knowledgeable about the ways their practice might be supported using knowledge-based techniques. We use design artifacts as a means to transfer knowledge and ideas back and forth between system builders and domain experts. Artifacts serve as concrete reference points (as "languages of doing"; Ehn, 1988, p. 58) facilitating communication of ideas between the design partners. The seed-building process is not a one-step affair. We are fortunate to have network domain experts at the University of Colorado (Nemeth, Snyder, & Seebass, 1989) who collaborate with us in the construction of the seed.

Figure 2. A portion of a network designer's logical map of the University of Colorado network.

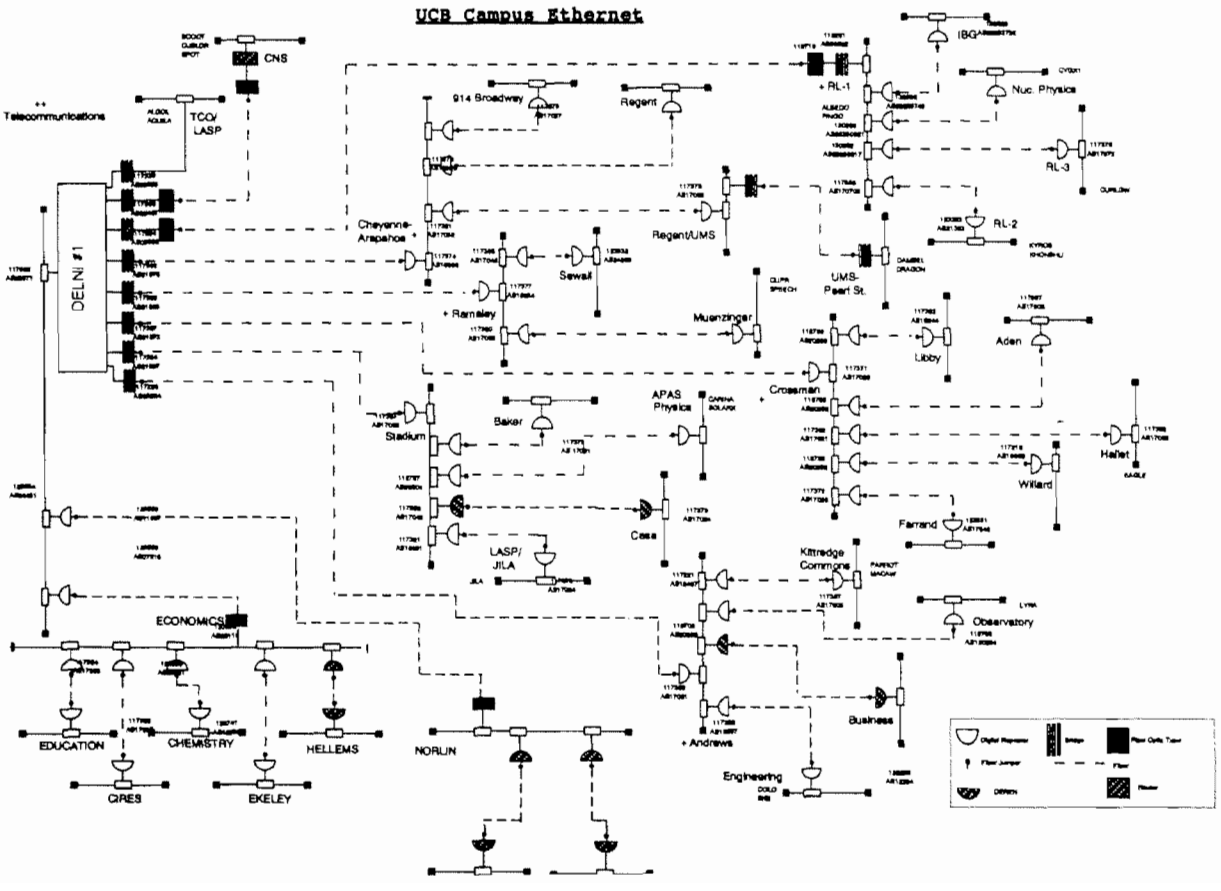


Figure 2 shows an example of a logical map used to ground discussions about changes to the real network. This map is a portion of a MacDraw document created by a network designer. This artifact has been used to focus issues that arose during design meetings. One can write on it directly, annotate it, and argue about it. It is easy to do end-user modification; one just adds a symbol in the lower right-hand-side box and then does a copy-and-paste onto the document. But this representation has the fundamental limitation that changes to this document have no semantic meaning beyond the interpretation given to them by the users. There are no computational mechanisms associated with the artifact.

Supporting the Evolution of Design Environments. No practical situation fits exactly into a preconceived category. Application domains and user requirements are constantly changing (Curtis, Krasner, & Iscoe, 1988). These changing environments require a design environment that is adaptable by the designer to fit unanticipated needs of the situation (Fischer & Girgensohn, 1990).

Design environments and individual design projects carried out within a design environment are complex systems. Complex systems are not only designed, but they need to evolve (as illustrated by Figure 1). Besides the goal of providing general domain information in the seed, a successful design environment seed needs to support the addition of new knowledge. This evolution of the general purpose seed to suit the individual design task and to support cooperation between designers is essential for the success of the design environment. In our application domain—digital communication networks—the technology is changing so rapidly that without evolution the information in the design environment would quickly become out of date.

The evolution of design environments will be severely limited if the domain experts are unable to incorporate new knowledge themselves. However, domain experts are in most cases unwilling to acquire detailed knowledge about programming and knowledge engineering. Therefore, mechanisms in support of end-user modifiability are required (Fischer & Girgensohn, 1990). End-user modifiability is important in design environments for the following reasons: (a) Competent practitioners usually know more than they can say; (b) tacit knowledge is triggered by situations and by breakdowns; (c) background assumptions cannot be completely articulated; (d) situations of practice are complex, unique, uncertain, conflicted, and unstable; and (e) initial moves must be reframed, because the changed situation most often deviates from the initial appreciation. The breakdowns are not experienced by the knowledge engineers but by the domain experts using the system. To support evolution on a continual basis, the people experiencing the breakdowns are in the best position to do something about them.

Because domain experts are interested not in evolving the seed per se but in

solving design problems, modifications to the seed must follow as a natural consequence of their work without requiring an unreasonable effort. Extensible critics and the addition of new palette items are examples of seed modifications (Fischer & Girgensohn, 1990).

The goal to create possibilities for domain experts to change systems provides a potential solution to address the maintenance problems in software design (CSTB, 1990), where maintenance characterizes a process of continued design and development rather than the traditional notion of dealing with wear and age. Seeded design environments provide a potential solution to the general and important problem of finding new ways to enhance existing software systems on a continual basis (Henderson & Kyng, 1991). The evolutionary growth of systems can be supplemented by *reseeding processes,* where knowledge engineers do major revisions of the systems.

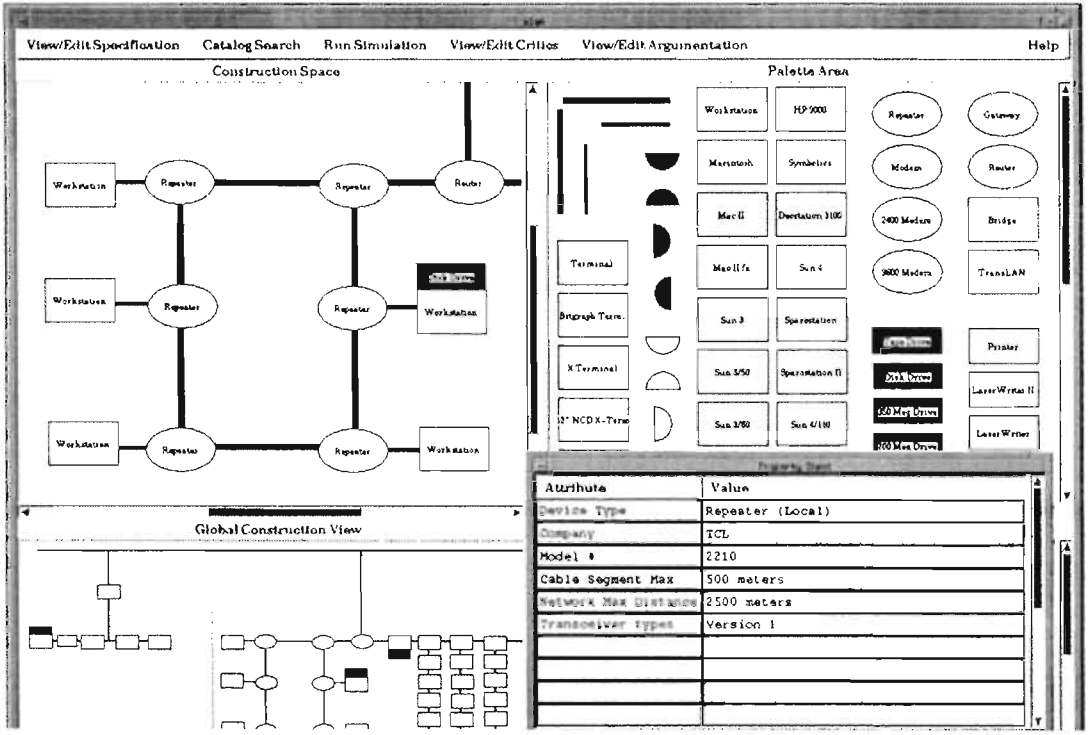
3.2. Application Domain

The application domain of digital communication networks meets several important criteria. The domain is of sufficient complexity to exhibit the kinds of problems we have described, and it is in some sense typical of a wide range of applications to which our work should be generalizable. Our previous work has concentrated on the domains of kitchen and user interface design (JANUS and FRAMER systems). These domains, especially the former, were chosen for a first feasibility study and do not show the generality of our approach.

A local area network links devices such as workstations, file servers, and printers using cables of different lengths and types. A partial construction kit for such networks is pictured in Figure 3. Networks can be viewed at different levels of abstraction. There is the level of individual devices and the level of subnets that are connected using bridges and gateways. At an even larger level, the local area network must be properly integrated into a nationwide network such as the Internet.

Designers of these networks must take into account criteria such as cost, reliability, and extensibility. There is a large body of design rules of different flexibility. Hard rules are rules that must be adhered to. For example, CSMA/CD networks must have a hierarchical topology and do not function in a ring topology. Some rules are bendable. For example, although technical specifications require that RS 232 asynchronous communication lines are no longer than 75 ft (22m), experience has shown that in normal circumstances they can safely be extended to several hundred feet (a few hundred meters). Soft rules are guidelines that describe good practice, and their violation may result in decreased performance and increased error rates in specific situations. Sources of design knowledge for this domain are books (Comer, 1988; Nemeth et al., 1989; Tanenbaum, 1981), standards such as IEEE 802.3, product literature, and local experts.

Figure 3. Construction kit for digital communication networks. The construction area shows a generic ring configuration in which machine types have not yet been decided or expressed. The palette area shows a number of devices from which the user can choose. Items in the palette vary in specificity to allow for a gradual expression of the design. The global construction view provides a higher level view of the construction space where the user can see how what is in the construction space relates to unseen parts of the network. A property sheet describing the attributes of a design object appears in the lower right corner of the screen.



Networks can rarely be designed as a whole, but they almost always evolve from a minimum configuration by the addition of needed connections to other networks and new hardware. This means that the design phase is never over, and decisions that were made long ago continue to interact with current decisions. The administration and design of local area networks often passes from network administrator to administrator. The new network administrator must often modify the design (to add new machines or to connect new rooms) of the previous administrator. Currently, this occurs with the new network manager not knowing the rationale of a previous design decision or what problems had come up in the past. This would not be the case if the former network manager's decisions were in the issue base and critics.

The design of networks often relies on a number of people who form the network support team and who each have to make decisions about the design in some way. Localized decisions (e.g., where and how to connect a new workstation to an existing network) should be made in the context of the overall design goals. Most networks are so large that no single person can comprehend all the details about the network design.

The design of digital communication networks is a discipline that, in the current information age, is in high demand in many sectors of society. Network design is typical of a wide range of other design domains. Scarcity of hard rules and conflicts among soft rules make an algorithmic problem-solving approach infeasible. Each solution illustrates tradeoffs between many goals such as cost and reliability. The constraints of the domain are ever changing as new hardware enters the market and new experience is gained. To achieve and to retain a high level of performance, designers must continuously learn and extend their knowledge accordingly.

4. A SCENARIO USING NETWORK

Figures 3, 4, and 5 are screen images of our prototype system illustrating how NETWORK provides for the gradual expression of a design, supports the situation talking back, uses critics to link the construction component with argumentation, and helps multiple designers notice conflicts in their collaborative work.

In our scenario, a network designer, "Tom," has been asked to create a subnet for a new research group and attach it to the rest of the engineering center's network. Figure 3 shows how Tom has chosen generic components from the palette to record an initial idea for the design because he does not yet know what types of workstations the research group will be using. By including generic devices in the palette as well as specific devices, the design environment provides for the gradual expression of a design over time. Figure 3 also shows the properties of a selected transceiver so that Tom can make sure it meets the needs of the current situation.

Figure 4. **Construction and argumentation. A user is notified of a conflict between the current design in the construction space and a recently added rule, which is described for the user in the critic window. The argumentation window provides rationale related to the rule.**

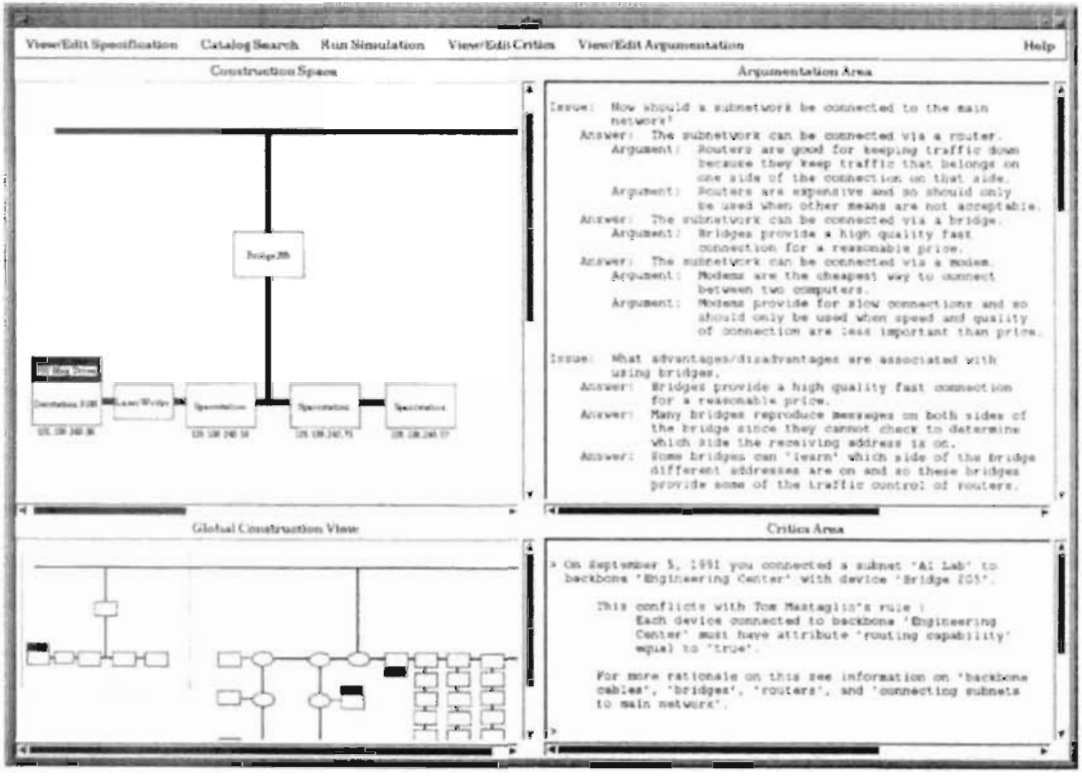
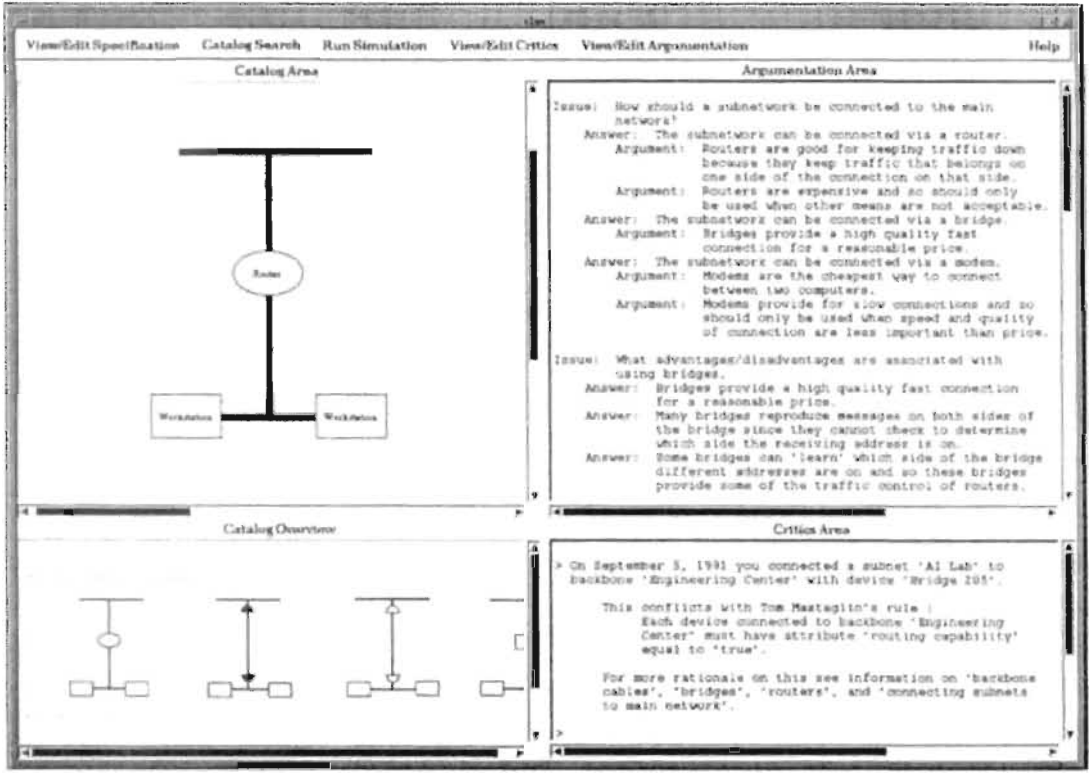


Figure 5. Catalog and argumentation. A user sees a prototypical use situation in the catalog area for possibilities described in the argumentation. The catalog overview shows several catalog examples that have been retrieved for the user.



Now, imagine that designer "Joe" used a bridge to connect a subnet to the main backbone a while ago. Later on in the project, Tom decides that backbone connections should be done only with "smart routers," and he writes a critiquing rule that enforces this. To illustrate a critic rule, this one can be represented as:

```
if connected(Device, "Engineering Center")
   then Routing-Capability(Device, "True")
```

When he activates this new rule, he is informed of current conflicts, in this case a bridge has previously been used in a way that violates the new rule. Tom is informed of this violation as soon as the critic rule is active. Figure 4 shows Joe's view the next time he logs onto the system. He is informed that a part of the design he handled is in conflict with a rule. The system knows that Joe should be informed because it keeps a design history: a log of when and by whom design units were placed. The critiquing component and the design history of the artifact support coordination between designers.

To continue with the scenario, Joe wants to look up rationale for bridges and routers, so he selects "connecting subnets to main networks" and sees the rationale related to that issue (see "Argumentation Area" in Figure 4). The critiquing component links the construction situation to the argumentation. Joe finds that Tom has added the rationale for routers to the argumentation base.

A continuation of this scenario shows how NETWORK uses actual and prototypical design artifacts to illustrate design rationale. Previously, Joe was informed that a design decision he made a while ago now conflicts with a new rule. To see a concrete illustration of the various answers to the issue of connecting subnets, he asks for an illustration of the answer "The subnetwork can be connected via a router" to the issue "How should a subnetwork be connected to the main network?" (see Figure 5). NETWORK orders the catalog of designs by their relevance to this issue and allows the designer to explore any of the catalog entries.

5. SYSTEM ARCHITECTURE AND IMPLEMENTATION

In this section, we present a system architecture that extends our previous work. We justify this architecture by showing how it addresses the problems identified in Section 2. The system architecture contains the following components (Fischer, Lemke, McCall, & Morch, 1991): a construction kit, argumentative hypermedia, a specification component, a simulation component, and a catalog. An important characteristic of the architecture is that design decisions made within the context of any given component affect the

information in each of the other components. In this way, the entire system continuously adjusts to the designer's actions, providing situated support for the design process.

5.1. Construction Kits

The construction kit is the principal medium for work in the integrated environment. It provides a palette of domain abstractions (see Figure 3) and supports the construction of artifacts using direct manipulation. The domain abstractions support human problem-domain communication (Fischer & Lemke, 1988) because they allow designers to think in terms of familiar domain abstractions. Construction kits support collaboration by providing a shared language for representing designs in a particular domain (Ehn, 1988; Resnick, 1991). As the domain building blocks are extended and refined, they are stored in the palette, where they are available to all users of NETWORK. The initial palette was modeled after design artifacts the network designers used. Certain icons were taken from network designers' documents (like the one shown in Figure 2), and others came from textbooks. The rapid change of technology in this domain confirmed the need for end-user modifiable palette items (Fischer & Girgensohn, 1990).

5.2. Argumentative Hypermedia

The issue base is a collection of argumentation on recurring issues in the problem domain that is structured according to the PHI serves relationship (McCall, 1991). Information fragments in the hypermedia issue base are linked according to what information serves to resolve an issue relevant to construction. The primary function of the issue base is to support construction by providing designers with information required to understand and resolve breakdown situations.

A design environment must provide mechanisms and tools to help designers quickly access the relevant information stored in the domain-oriented issue base. In the construction situation, argumentative information is accessible from objects displayed on the screen, such as palette items (McCall et al., 1990). The internal representation for each object includes the location of related information in the issue base. The system performs the task of locating information for the designer.

Issue-based argumentation provides a forum and repository for an ongoing discourse about principles and recurring themes in design, allowing designers to communicate indirectly and thereby supporting collaboration. The issue base has the following functions: (a) supporting reflection in action by explaining breakdowns (identified by critic messages; Fischer, Lemke,

Mastaglio, & Morch, 1991) and suggesting ways to repair them; (b) allowing designers to record design rationale and to react to rationale supplied by others (Fischer, Lemke, McCall, & Morch, 1991); and (c) making designers aware of issues, possible answers, and argumentation as they browse the issue base. For the development of the seed of our issue base, we videotaped a number of design sessions. The videotapes showed that deliberation of issues is a major activity in design—for a detailed analysis of design activities, see Olson, Olson, Carter, and Storrøsten (1992)—and that most argumentation arises in reference to a specific situation and thus is expressed in terms of that situation. Constructing the issue base seed will require transforming this argumentation into a general form and assembling the various argumentation fragments into a coherent PHI structure. This work demands domain expertise as well as experience in PHI methodology. A seed generated by experts will save designers the cognitive effort of providing a base structure to the issue base.

5.3. Specification Component

Designers must trade off competing goals such as minimal cost, reliability, and extensibility. With a specification component, designers can input characteristics of the task, and the system can use them to tailor its information structures by filtering out argumentation, critics, and catalog examples that are not relevant to the specified task.

Specifications are not completely known before beginning a design project. Instead, they are incrementally refined throughout the design process. By allowing incremental definition of specifications, design environments support the integration of problem setting and problem solving (Rittel, 1984) as well as the coevolution of specifications and implementations (Swartout & Balzer, 1982). In addition, specifications have side effects that conflict with the current state of the design in subtle ways. NETWORK incorporates work done on JANUS that allows the system to detect such specification interactions and notify the designer by causing the situation to talk back (Fischer & Nakakoji, 1991). This way, designers can quickly assess tradeoffs and recognize specification tradeoffs. Active, adaptive issue bases filter out irrelevant information, resulting in a smaller information space having a larger percentage of information relevant to the specific situation (McCall, Ostwald, & Shipman, 1991).

Specifications provide high-level guidelines for both designers and the environments supporting design. In collaborative design, specifications serve to help coordinate the work of group members by providing a common framework in which to operate. For example, if a network is specified to support the Ethernet protocol, then group members must design accordingly

and trust others to do the same. Tools to help understand the side effects of a specification modification are important in group design because designers cannot know an entire project in enough detail to predict how a specification will affect the work of others. Design environments use partial specifications as a resource for situating information structures. Specifications are also used by the system to classify designs. Designs stored by the system may be retrieved based on their specifications. This use of specifications is discussed in Section 5.6 in relation to CATALOG EXPLORER.

5.4. Catalog

The catalog (see Figure 5), a collection of predesigned artifacts, illustrates the space of possible designs in the domain. It serves as the repository of designs constructed by the group over time. The catalog supports design by modification, a copy-and-edit strategy. Catalog items may be copied into the construction situation, where they are modified to fit the requirements of a particular design situation. Reusing successful designs, or portions of designs, saves time and cognitive effort and provides stable components for constructing complex designs. Designers can assess the relevance of the example to the situation at hand by using critics and the design rationale associated with the example.

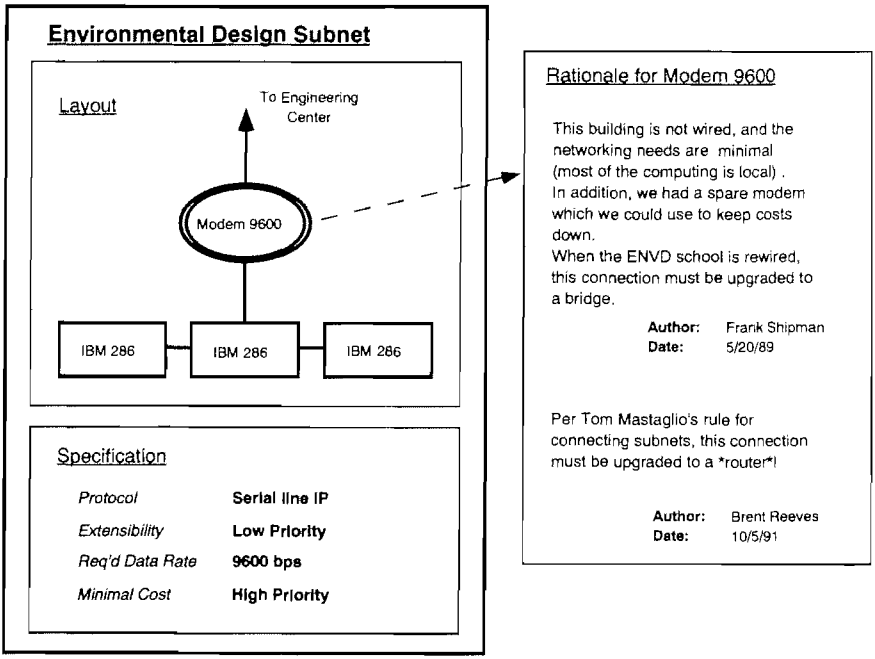
The catalog supports collaborative design by storing completed designs for future use, along with their rationale and specification (see Figure 6). Designers concerned with maintenance need this information to understand the original design and to make informed decisions in the future.

5.5. Simulation Component

Evaluations of the JANUS system by expert designers (Fischer et al., 1989a) have demonstrated a need for simulating usage scenarios (Ehn, 1988; Fischer & Reeves, 1992) with the artifact being designed. Such functional simulations can take the form of deterministic mathematical models as well as informal what-if games. Functional simulation improves the capability of the construction situation to talk back to the designer. Work in the domain of voice dialog systems suggests that simulations enhance a designer's ability to understand how modifications affect the designed artifact's behavior in use situations (Sumner, Davies, Lemke, & Polson, 1991). Information provided by simulation complements the argumentative component, which can never capture all relevant aspects of use situations (Suchman, 1987).

In the domain of communication networks, where many performance measures require substantial computation, the inclusion of a simulation component is a necessity. From our videotaped design sessions, it became

Figure 6. **Schematic of a design representation in the catalog. Catalog entries contain a graphical representation, specification, and design rationale. This example shows the catalog representation of the college of environmental design's subnetwork, including the rationale for a project-specific issue.**



apparent that one use of example designs, as found in the catalog, would be the running of simulations on slightly modified example designs. By doing so, designers can find the effects of such modifications on a completed design, and differences in opinion can be reduced.

The purpose of our simulation component is not to calculate detailed numerical values over an entire network. Rather, we want to estimate local effects associated with different usage scenarios for a given segment of a network. An example of a useful simulation is to test the load on a crucial network component, such as a file server having a known performance threshold, given different configurations, hardware platforms, and software requirements in a subnet. This type of information is essential for the enforcement of performance specifications.

With the help of a simulation component, breakdowns can be identified that are invisible in static construction kits. General design principles, such as answers and arguments in the argumentation component, cannot take into account the variety of real-life contingencies that characterize design in

complex domains. Even relatively simple simulations can provide information to the designer that helps in evaluating design principles in light of unanticipated design situations.

5.6. Links Between the Components in HYDRA

HYDRA derives its power from the integration of its components. At each stage in the design process, the partial design embedded in the design environment is a "situation that can talk back" to users by suggesting what they should attend to next. This direction to new subgoals permits new information to be extracted from memory and reference sources and is another step to be taken toward the development of the design. The primary mechanisms for integration in HYDRA are CONSTRUCTION ANALYZER, CATALOG EXPLORER, and ARGUMENTATION ILLUSTRATOR.

CONSTRUCTION ANALYZER is a critiquing system (Fischer, Lemke, Mastaglio, & Morch, 1991) in which each critic is responsible for recognizing a specific construction situation representing a potential breakdown. For example, in the scenario discussed in Section 4, one critic is responsible for detecting backbone connectors that do not use smart routing devices. The critic mechanism consists of a feature detector, a mouse-sensitive message, and a location in the issue base where relevant information is stored. If the situation is detected, the critic's message is displayed in the critics area of the screen. If the designer chooses to click on the message, the argumentation window is opened, and information located by the critic is displayed. The designer may also choose to ignore the critic message and proceed with the construction process.

CONSTRUCTION ANALYZER integrates construction and argumentation, enabling the system to play an active role in providing information that becomes relevant as the construction situation changes.

CATALOG EXPLORER (Fischer & Nakakoji, 1991, 1992) helps users to search the catalog space according to the task at hand. It retrieves design examples similar to the current construction situation and orders sets of examples by their appropriateness to the current specification. CATALOG EXPLORER links the specification and construction components with the catalog. It exploits the information articulated in a partial specification or a partial construction to prioritize the designs stored in the catalog with respect to the task at hand.

Design objects stored in a catalog can be reused for case-based reasoning, such as providing a solution to a new problem, evaluating and justifying decisions behind the partial construction or specification, and informing designers of possible failures. Case-based reasoning (Riesbeck & Schank, 1989) complements generalized argumentative reasoning when principles are not sufficiently well defined.

ARGUMENTATION ILLUSTRATOR (Fischer, Lemke, McCall, & Morch, 1991) helps users to understand the information given in the argumentation component by using a catalog design example as a source of concrete realization. Explanations given as argumentation are often highly abstract and conceptual. Concrete design examples that match explanations help users to understand abstract concepts by showing their use in specific examples.

The set of catalog examples that illustrates an argumentative concept is computed by applying a critic rule to each example. In the scenario, the user asked for an illustration of the answer, "The subnetwork can be connected via a router," to the issue, "How should a subnetwork be connected to the main network?" In this case, a critic that recognizes "subnetworks connected to main networks by a router" is applied to each example design in the catalog to compute the set of relevant examples. It is important to find examples that are similar to the current design situation so that designers are able to understand how the general principle is instantiated in the example design and how the same principle might apply to their design situation.

6. FUTURE WORK

Reuse of Design Rationale. A design rationale is a large additional product of the design process (Fischer, Lemke, McCall, & Morch, 1991). Creating and representing a design rationale is a great effort. Reuse of existing issue bases has the potential to dramatically reduce this effort. Just as reuse and redesign is an indispensable goal in programming (and has become more feasible in object-oriented design methodologies), reuse of design rationale is equally important. Every project is unique in some respects, and few if any projects are unique in all respects. Therefore, the contents of a project issue base are not entirely unique to that project. Similar projects overlap substantially in issues, answers, and arguments. This is not to say, however, that the issues are resolved in the same way, but merely that a great deal of the reasoning is shared by projects.

Reusable issue bases can serve as a seed that grows with each new design project. Each project extends and enhances the reusable issue base. The issue base being reused provides information about how to decompose the task, possible answers to issues, and principles of design. Domain-oriented issue bases amplify the designer's ability to reflect on issues. Recurring design issues can be researched intensively, and results of this can be stored at the appropriate location in the issue base for use by designers encountering similar decisions in the future. This allows the "folk theories" of designers to be subjected to rigorous scientific scrutiny.

Cumulative domain-oriented issue bases could also foster communication among designers, researchers, and users about recurring matters of design. The PHI subissue relationship is crucial to making issue bases reusable

(McCall, 1991). The hierarchical grouping of issues allows argumentation systems to be built that filter issue bases according to the specifics of the new task.

Evolution of Design Environments. Design environments are a repository for design information in a certain domain. Accumulation and modification of knowledge of the design environment should be regarded as evolution of a design environment. There are two types of evolution: *ontogeny* and *phylogeny*—terms used in biology. Ontogeny means development of an individual organism, and phylogeny refers to history of a species.

Ontogenic evolution of a design environment means that, as a design process proceeds, a designer's task at hand is articulated, and the content and representation of the knowledge of the environment are reframed and restructured according to the evolving task at hand. Phylogenic evolution of a design environment means that designers (i.e., domain experts) add their knowledge to the system within their working context by constantly using the design environment. Through ontogenic interaction, the design environment's information structures are restructured to contain information relevant to the task at hand. In contrast, through phylogenic interaction, the design environments come closer to achieving coverage of their target domain.

In our future work, we will investigate in detail how ontogenic and phylogenic processes interact with each other, that is, under which circumstances knowledge should remain with individual design projects and under which circumstances it should become a part of the general design environment.

Addressing the Challenges of Designing CSCW Applications. As our system integrates support for individual use into CSCW design and development contexts, it will encounter the challenges faced by CSCW applications in general (Grudin, in press). We are addressing many of these challenges from the outset.

A major source of failure of CSCW applications is that key users do not perceive direct benefit from the use of the system, yet they must do work to support it. Our initial focus on support for individuals has required us to provide benefits for each user. In NETWORK, a major initial effort is required to provide domain seed knowledge. Once the seed is in place, the reuse of this material provides benefits to all users. The ratio of reuse to initial construction effort is likely to be a key determinant of the system's success in a given domain. We expect that design environments will be less useful in novel design circumstances (e.g., building a research prototype) than in more repetitive design situations, such as designing a new kitchen or a new network. The seed also circumvents the related "critical mass" problem that

afflicts CSCW applications. For example, if a group had to build a catalog from scratch, the labor required of all members before benefits from reuse appeared might be such that few would persist in the use of the system, and it would never become viable. But the seed, and its evolution through use, supports a large number of knowledge enterers and thereby provides a critical mass situation at the outset.

As noted earlier, by focusing on shared information systems, we allow users to define their own relationships to the system and avoid conflicts arising in variations or exceptions in roles and procedures within a group. Tailorability provides support for variability. By starting with individual use situations in mind, we hope to avoid another CSCW problem—the tendency to provide only very infrequently used communication and coordination features that are less readily learned and recalled by users and less intuitively understood by developers.

Two CSCW challenges we must explore carefully are the limitations of conscious, rational decision making and the difficulty of anticipating problems that will arise when a system is introduced in a workplace. Social and motivational issues that arise in group contexts are often handled through tacit or unconscious mechanisms that resist the explicit treatment inherent to knowledge bases. Tools that contribute to decision making can be threatening to people who successfully use other approaches to influence decision making.

7. CONCLUSIONS

We have presented a methodology and system architecture for supporting group design that allows individuals to be informed about how their work interacts with the work of other group members. Designers should be informed on a need-to-know basis, leaving them free to concentrate on their design task. We are developing knowledge-based mechanisms to identify situations in which artifacts created by individuals and group design goals interact. These mechanisms then alert designers of the interactions. Systems with such capabilities will reduce the cognitive effort required for individuals to design asynchronously, but in coordination, with other members of a group.

Designers operate in the context of an integrated design environment, such as NETWORK, which is based on the domain-independent architecture HYDRA. A fundamental aspect of this work is the concept of a group memory containing domain knowledge ranging from general design principles to information about specific projects and their associated design rationale. The group memory allows design knowledge to be useful beyond the life of a single project and to survive changes in personnel. To reduce the effort to create a group memory, a design environment must supply a seed for such a group memory. The seed provides general domain knowledge and structure for the

group memory. Our work investigates the requirements for a seed and how the seed can evolve through time as the design environment is used. As the group memory becomes large, techniques of finding and presenting information relevant to a specific design become important.

We have described the domain of network design. This domain undergoes continuous change and requires a great deal of design expertise not found in textbooks. Network design and administration involves groups of people and extends over time. The network design and administration task is passed from person to person. Modifications made to the network must take into account the rationale for past design decisions as well as decisions made by other members of the design team. Computers can support this task through a combination of a group memory and mechanisms for alerting the designer to information relevant to the current task.

Acknowledgments. We thank the members of the Human-Computer Communication group at the University of Colorado, who contributed to the conceptual framework and the systems discussed in this article. We are also grateful to Hal Eden, Ken Klingenstein, Evi Nemeth, and David Wood, who provided expertise in the network design domain.

Support. The research was supported by the National Science Foundation under Grant No. IRI-9015441 and by the NYNEX Science and Technology Center.

REFERENCES

Akscyn, R. M., McCracken, D. L., & Yoder, E. A. (1988). KMS: A distributed hypermedia system for managing knowledge in organizations. *Communications of the ACM, 31*(7), 820-835.

Bullen, C. V., & Bennett, J. L. (1990). Learning from user experience with groupware. *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW '90),* 291-302. New York: ACM.

Comer, D. E. (1988). *Internetworking with TCP/IP: Principles, protocols, and architecture.* Englewood Cliffs, NJ: Prentice-Hall.

Computer Science and Technology Board. (1990). Scaling up: A research agenda for software engineering. *Communications of the ACM, 33*(3), 281-293.

Conklin, J., & Begeman, M. (1988). gIBIS: A hypertext tool for exploratory policy discussion. *Transactions of Office Information Systems, 6*(4), 303-331.

Curtis, B., Krasner, H., & Iscoe, N. (1988). A field study of the software design process for large systems. *Communications of the ACM, 31*(11), 1268-1287.

Draper, S. W. (1984). The nature of expertise in UNIX. *Proceedings of INTERACT '84, IFIP Conference on Human-Computer Interaction,* 182-186. Amsterdam: Elsevier Science.

Ehn, P. (1988). *Work-oriented design of computer artifacts.* Stockholm: Almquist & Wiksell.

Ellis, C. (1991). Models of computer supported cooperative work. In P. Kerola, R. Lee, K. Lyytinen, & R. Stamper (Eds.), *Collaborative work, social communications and information systems* (pp. 373-385). Amsterdam: North-Holland.

Ellis, C. A., Gibbs, S. J., & Rein, G. L. (1991). Groupware: Some issues and experiences. *Communications of the ACM, 34*(1), 38-58.

Erickson, T. D. (1989). Interfaces for cooperative work: An eclectic look at CSCW '88. *SIGCHI Bulletin, 21*(1), 56-64.

Fischer, G. (1988). Enhancing incremental learning processes with knowledge-based systems. In H. Mandl & A. Lesgold (Eds.), *Learning issues for intelligent tutoring systems* (pp. 138-163). New York: Springer-Verlag.

Fischer, G., & Girgensohn, A. (1990). End-user modifiability in design environments. *Proceedings of the CHI '90 Conference on Human Factors in Computing Systems,* 183-191. New York: ACM.

Fischer, G., & Lemke, A. C. (1988). Construction kits and design environments: Steps toward human problem-domain communication. *Human-Computer Interaction, 3,* 179-222.

Fischer, G., Lemke, A. C., Mastaglio, T., & Morch, A. (1991). The role of critiquing in cooperative problem solving. *ACM Transactions on Information Systems, 9*(2), 123-151.

Fischer, G., Lemke, A. C., McCall, R., & Morch, A. (1991). Making argumentation serve design. *Human-Computer Interaction, 6,* 393-419.

Fischer, G., McCall, R., & Morch, A. (1989a). Design environments for constructive and argumentative design. *Proceedings of the CHI '89 Conference on Human Factors in Computing Systems,* 269-275. New York: ACM.

Fischer, G., McCall, R., & Morch, A. (1989b). JANUS: Integrating hypertext with a knowledge-based design environment. *Proceedings of Hypertext '89,* 105-117. New York: ACM.

Fischer, G., & Nakakoji, K. (1991). Making design objects relevant to the task at hand. *Proceedings of AAAI-91, Ninth National Conference on Artificial Intelligence,* 67-73. Cambridge, MA: AAAI Press/MIT Press.

Fischer, G., & Nakakoji, K. (1992). Beyond the macho approach of artificial intelligence: Empower human designers—Do not replace them. *Knowledge-Based Systems Journal, 5,* 15-30.

Fischer, G., & Reeves, B. N. (1992). Beyond intelligent interfaces: Exploring, analyzing and creating success models of cooperative problem solving. *Applied Intelligence, 1,* 311-332.

Gero, J. S. (1990). A locus for knowledge-based systems in CAAD education. In M. McCullough, W. J. Mitchell, & P. Purcell (Eds.), *The electronic design studio* (pp. 49-60). Cambridge, MA: MIT Press.

Greif, I. (Ed.). (1988). *Computer-supported cooperative work: A book of readings.* San Mateo, CA: Morgan Kaufmann.

Grudin, J. (1988). Why CSCW applications fail: Problems in the design and evaluation of organizational interfaces. *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW '88),* 85-93. New York: ACM.

Grudin, J. (1990). Groupware and cooperative work: Problems and prospects. In B. Laurel (Ed.), *The art of human-computer interface design* (pp. 171-185). Reading, MA: Addison-Wesley.

Grudin, J. (in press). Groupware and social dynamics: Eight challenges for developers. *CACM.*

Halasz, F. G. (1988). Reflections on notecards: Seven issues for the next generation of hypermedia systems. *Communications of the ACM, 31*(7), 836-852.

Henderson, A., & Kyng, M. (1991). There's no place like home: Continuing design in use. In J. Greenbaum & M. Kyng (Eds.), *Design at work: Cooperative design of computer systems* (pp. 219–240). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Johansen, R. (1988). *Groupware: Computer support for business teams.* New York: Free Press.

Lee, J. (1990). SIBYL: A tool for managing group decision rationale. *Proceedings of the Conference on Computer-Supported Cooperative Work,* 79–92. New York: ACM.

Lemke, A. C., & Fischer, G. (1990). A cooperative problem solving system for user interface design. *Proceedings of AAAI–90, Eighth National Conference on Artificial Intelligence,* 479–484. Cambridge, MA: AAAI Press/MIT Press.

Lenat, D., & Guha, R. V. (1990). *Building large knowledge-based systems.* Reading, MA: Addison-Wesley.

Lynch, K. J., Snyder, J. M., Vogel, D. R., & McHenry, W. K. (1990). The Arizona analyst information system: Supporting collaborative research on international technological trends. In S. Gibbs & A. A. Verrijn-Stuart (Eds.), *Multi-user interfaces and applications* (pp. 159–174). Amsterdam: North-Holland.

Malone, T. W., Grant, K. R., Lai, K.-Y., Rao, R., & Rosenblitt, D. (1986). Semi-structured messages are surprisingly useful for computer-supported coordination. *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW '86),* 102–114. Austin, TX: MCC.

Malone, T. W., Grant, K. R., Lai, K.-Y., Rao, R., & Rosenblitt, D. (1988). Object lens: A "spreadsheet" for cooperative work. *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW '88),* 115–124. New York: ACM.

McCall, R. (1989). MIKROPLIS: A hypertext system for design. *Design Studies, 10*(4), 228–238.

McCall, R. (1991). PHI: A conceptual foundation for design hypermedia. *Design Studies, 12*(1).

McCall, R., Bennett, P., d'Oronzio, P., Ostwald, J., Shipman, F., & Wallace, N. (1990). PHIDIAS: Integrating CAD graphics into dynamic hypertext. *European Conference on Hypertext (ECHT '90),* 152–165. Cambridge: Cambridge University Press.

McCall, R., Mistrik, I., & Schuler, W. (1981). An integrated information and communication system for problem solving. *Proceedings of the Seventh International CODATA Conference,* 107–113. London: Pergamon.

McCall, R., Ostwald, J., & Shipman, F. (1991). Supporting designers' access to information through virtually structured hypermedia. *Proceedings of the 1991 Conference on Intelligent Computer Aided Design,* 116–127. New York: Elsevier.

McDermott, J. (1982). R1: A rule-based configurer of computer systems. *Artificial Intelligence, 19,* 39–88.

Nemeth, E., Snyder, G., & Seebass, S. (1989). *Unix system administration handbook.* Englewood Cliffs, NJ: Prentice-Hall.

Olson, G. M., Olson, J. S., Carter, M. R., & Storrøsten, M. (1992). Small group design meetings: An analysis of collaboration. *Human–Computer Interaction, 7*(4).

Resnick, L. B. (1991). Shared cognition: Thinking as social practice. In L. B. Resnick, J. M. Levine, & S. D. Teasley (Eds.), *Perspectives on socially shared cognition* (pp. 1–20). Washington, DC: American Psychological Association.

Riesbeck, C., & Schank, R. C. (1989). *Inside case-based reasoning.* Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Rittel, H. W. J. (1972). On the planning crisis: Systems analysis of the first and second generations. *Bedriftsokonomen, 8,* 390-396.

Rittel, H. W. J. (1984). Second-generation design methods. In N. Cross (Ed.), *Developments in design methodology* (pp. 317-327). New York: Wiley.

Rittel, H. W. J., & Webber, M. M. (1984). Planning problems are wicked problems. In N. Cross (Ed.), *Developments in design methodology* (pp. 135-144). New York: Wiley.

Schön, D. A. (1983). *The reflective practitioner: How professionals think in action.* New York: Basic Books.

Shipman, F., Chaney, R., & Gorry, T. (1989). Distributed hypertext for collaborative research: The virtual notebook system. *Proceedings of Hypertext '89,* 129-135. New York: ACM.

Simon, H. A. (1981). *The sciences of the artificial.* Cambridge, MA: MIT Press.

Sorgaard, P. (1988). A framework for computer supported cooperative work. In J. Kaasboll (Ed.), *Report of the 11th informations systems research seminar in Scandinavia* (pp. 620-640). Oslo: University of Oslo, Department of Informatics.

Steele, R. L. (1987). An expert system application in semicustom VLSI design. *Proceedings of the 24th IEEE/ACM Design Automation Conference,* 679-686. Los Angeles: IEEE Computer Society Press.

Stefik, M., Foster, G., Bobrow, D. G., Kahn, K., Lanning, S., & Suchman, L. (1987). Beyond the chalkboard: Computer support for collaboration and problem solving in meetings. *Communications of the ACM, 30*(1), 32-47.

Suchman, L. A. (1987). *Plans and situated actions.* Cambridge, England: Cambridge University Press.

Sumner, T., Davies, S., Lemke, A. C., & Polson, P. (1991). *Iterative design of a voice dialog design environment* (Tech. Rep. No. CU-C5-546-91). Boulder: University of Colorado, Department of Computer Science.

Swartout, W. R., & Balzer, R. (1982). On the inevitable intertwining of specification and implementation. *Communications of the ACM, 25*(7), 438-439.

Tanenbaum, A. (1981). *Computer networks: Toward distributed processing systems.* Englewood Cliffs, NJ: Prentice-Hall.

Walker, J. H. (1987). Document examiner: Delivery interface for hypertext documents. *Hypertext '87 Papers,* 307-323. Chapel Hill: University of North Carolina.

Winograd, T. (1988). A language/action perspective on the design of cooperative work. *Human-Computer Interaction, 3,* 3-30.

Winograd, T., & Flores, F. (1986). *Understanding computers and cognition: A new foundation for design.* Norwood, NJ: Ablex.

Yakemovic, K. C., & Conklin, E. J. (1990). Report of a development project use of an issue-based information system. *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW '90),* 105-118. New York: ACM.

HCI Editorial Record. First manuscript received April 15, 1991. Revisions received October 4, 1991, and April 6, 1992. Accepted by Gary Olson. -- *Editor*
