Journal of Applied Intelligence 1, 311–332 (1992)

Beyond Intelligent Interfaces: Exploring, Analyzing, and Creating Success Models of Cooperative Problem Solving

GERHARD FISCHER & BRENT REEVES

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

Received March 1991, Revised August 1991

Abstract. Cooperative problem-solving systems are computer-based systems that augment a person's ability to create, reflect, design, decide, and reason. Our work focuses on supporting cooperative problem solving in the context of high-functionality computer systems. We show how the conceptual framework behind a given system determines crucial aspects of the system's behavior. Several systems are described that attempted to address specific shortcomings of prevailing assumptions, resulting in a new conceptual framework. To further test this resulting framework, we conducted an empirical study of a success model of cooperative problem solving between people in a large hardware store. The conceptual framework is instantiated in a number of new system-building efforts, which are described and discussed.

Key words: Success model, knowledge-based systems, cooperative problem solving, intelligent interfaces, empirical studies, integrating problem setting and problem solving, shared understanding, high-functionality systems.

1. Introduction

We explore conceptual frameworks, methodologies, and technologies to develop cooperative problem-solving systems and exploit the unique opportunity offered by powerful computer systems. The purpose is to augment human potential and productivity [1–3], and not to replace humans with automated systems.

Our research approach, and the structure of this paper, is illustrated in Figure 1. We first review work that has been done in *cooperative problem-solving systems* and discuss why they provide a better conceptual framework for joint human-computer systems than intelligent interfaces. Shortcomings of these systems motivated us to look for success models [4–6] and alternative conceptual frameworks (such as situated cognition approaches [7–10]). The major portion of this paper addresses how success models help to confirm intuitions and introduce new challenges. We describe *integrated, domain-oriented, knowledge-based design* environments as prototypes of a second generation of cooperative problem-solving systems. From these sources, we draw lessons for a new conceptual framework for joint human computer systems.

2. First Generation of Cooperative Problem-Solving Systems

This section describes several issues that have surfaced in research on cooperative problem-solving systems. First, an analysis is made of different conceptual frameworks for integrating user interfaces and knowledge-based systems. Next, several dimensions of cooperative prob-

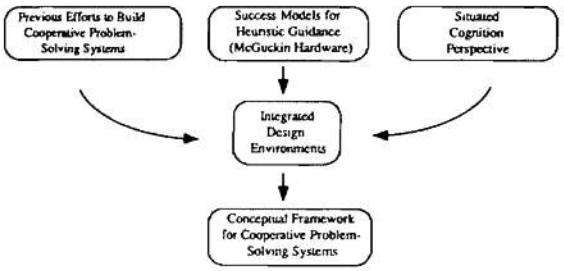

Fig. 1. Research approach.

The basic approach taken in the research described in this paper. By analyzing previous shortcomings and looking to existing success models in domains other than computer science, and then placing the lessons learned into the larger context of situated cognition research, we are building integrated design environments, which help us to incrementally refine an evolving conceptual framework for cooperative problem-solving systems.

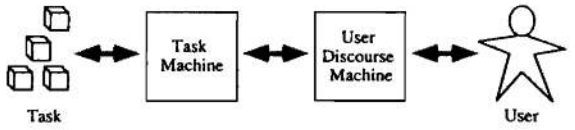

Fig. 2. Intelligent interfaces vs. interfaces to intelligent systems.

A simplification of Card's [12] Triple Agent Model of Human-Computer Interaction (which, in turn was inspired by Sheridan, Fischhoff, Posner, and Pew, 1983, Fig. 4-1). Card used the original figure to illustrate the different perspectives of three agents: User, Task machine, and User Discourse Machine. We use this simplified version to show how Intelligent Interfaces focus on the Task machine, whereas Interfaces to Intelligent Systems focus on the User Discourse Machine.

lem-solving systems are discussed, followed by a brief description of earlier prototypes and how they fell short of being truly cooperative problem-solving systems. We conclude by arguing that systems need to be both usable and useful [11], leading to high-functionality systems.

2.1. Interfaces to Intelligent Systems and Intelligent Interfaces

Traditionally the Artificial Intelligence community has classified user interface research into two subareas: "interfaces to intelligent systems" and "intelligent interfaces." Although these terms have been used mostly without any effort to define them, we will use a classification effort (inspired by a model from [12]; see Figure 2) to clarify how these terms may be defined.

Intelligent interfaces can now be defined by an attempt to put intelligence into the user discourse machine. The WEST system [13] can be considered an example of an intelligent interface. The underlying problem domain (computing algebraic expressions to satisfy certain objectives) is rather simple, but the user discourse machine of WEST consists of a number of interesting components such as a user modelling component, an explanation component, and a tutoring component.

Alternatively interfaces to intelligent systems are an attempt to put intelligence into the task machine. MYCIN [14] is an example of such a system. Although there has been an effort in MYCIN to put some intelligence into the user discourse machine (e.g., to support explanations [15]), these efforts have been modest compared to that of modelling the task.

The separating of the interface from the underlying application is inadequate for many system-building efforts. We support this claim with a human analogy: a person who can communicate well but knows very little has severe limitations as a cooperative partner, just as a person who knows a lot but cannot communicate. Cooperative problem-solving systems are an attempt to avoid this separation and increase the usefulness and usability by a tight integration of interaction mechanisms with the underlying domain knowledge.

2.2. Dimensions of Cooperative Problem-Solving Systems

Our original system-building efforts were very much influenced by some of the major shortcomings of expert systems. The major difference between classical expert systems (such as MYCIN [14] and RI [16]) and cooperative problem-solving systems is that the human is much more an active agent and participant in the latter. Tradi-

tional 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. Thus different role distributions may be chosen depending on the user's knowledge, the user's goals and the task domain. A cooperative system requires much richer communication facilities than those offered by traditional expert systems.

The following issues are important dimensions of research in cooperative problem-solving systems:

Understanding complex task domains. The interaction paradigms for dealing with complex information stores 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 precise articulation of a task is *the* most difficult problem [17]. Users of such systems suffer from a lack of knowledge about the interdependencies between problem setting and solving, and they do not know 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; therefore specifications must be constructed incrementally.

The level of cooperation between human and computer. Cooperative problem solving systems consisting of a human and a computer can exploit the asymmetry of the two communication partners. Humans contribute what they 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.) [18].

The impact of communication breakdowns. Effective assistance depends on a collaborative effort in which advisor and client work together to detect and repair troubles that arise. In cooperative problem-solving systems, breakdowns are not as detrimental as in expert systems, because humans are part of the overall system and can step in if necessary. 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 that when these inevitable troubles do arise, there are not the same resources available for their detection and repair"* [8]. A cooperative agent needs to understand the nature of open problems, the intentions of the problem solver, and the fact that goals are modified during the problem-solving process.

The role of background assumptions. We need a better understanding of the possibilities and limitations of expert systems research. We have to define the characteristics for problems that 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 [9]. The nature of expertise lies not only in solving a problem or explaining the results (which some expert systems can do to some extent), but in learning incrementally and restructuring one's knowledge, in breaking rules, in determining the relevance of something, and in degrading gracefully if a problem is not within the core of the expertise. Knowledge-based systems should be built on the premise that background assumptions can never be fully articulated.

Semi-formal versus formal approaches. Semi-formal systems [19–20] do not require the computer to interpret all information structures, but just to serve as a delivery system of information to be read and interpreted by people. Semi-formal systems 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." Humans often enjoy the process and not just the final product; they want to take part in something. This is why they build model trains, plan their vacations, and design their own kitchens. Automation is a two-edged sword. At one extreme, it is a servant, relieving humans of the tedium of low-level operations and freeing them for higher cognitive functions. Many people do not enjoy checking documents for spelling errors, and they

welcome the automation provided by spelling checkers in word processors. At the other extreme, automation can reduce the status of humans to "button pushers" and can strip their work of its meaning and satisfaction. The challenge is to automate tasks that people consider tedious or uninteresting, but these change as technology changes.

2.3. *Brief Discussion of Our Earlier Prototypes*

Many knowledge-based systems are built based on some of the following assumptions: (1) users of these systems can fully articulate their problems in advance, (2) users will ask for help, (3) a consultation model of interaction (in which users serve mostly as data sources) is behaviorally acceptable, and (4) general purpose programming environments are sufficient for supporting cooperative problem solving [3]. We believe these assumptions are *unfounded*.

The assumption that users can fully articulate problems in advance has been refuted in several studies [10]. Curtis, Krasner, and Iscoe [21] observed in an empirical study of large software projects: "*Even when a customized system was developed for one client, the requirements often provided a moving target for designers. During system development, the customer, as well as the developer, learned about the application domain.*" Many current software development methodologies (such as the waterfall model) falsely assume that problems are well defined.

The assumption that users are always capable of asking for help breaks down as soon as the system becomes very complex. Users are unable to ask about information they do not know exists.

MYCIN [14] is an example of a system that was based on the assumption that human-computer interaction is well supported by a consultation model in which the computer asks the human questions. From an engineering point of view, MYCIN had the advantage of being clear and simple: the program controlled the dialogue. But empirical studies have shown that these programs are behaviorally unacceptable [22].

General purpose tools are fundamentally limiting because the solution space represented by them is too far away from the problem space. In order to bridge the gap between general purpose tools and complex problem-solving environments, we need stable subsystems at various levels in between. Complex systems develop faster if they can build on stable subsystems [23] and if they can be based on a marketplace of developed pieces of knowledge [2].

In order to overcome some of these conceptual deficiencies, we have previously built a number of prototype systems (this brief annotated list is restricted to our own efforts; other research groups have addressed these problems as well.)

- *HELGON: Incremental Construction of Queries by Reformulation.* HELGON [24] is based on the retrieval-by-reformulation paradigm [25], which was derived from a theory of human remembering. This theory postulates that humans incrementally construct queries and naturally think about categories of things in terms of specific examples. HELGON supports the incremental description of a desired object with multiple specification techniques.
- *LISP-CRITIC AND ACTIVIST: Critiquing Users' Work and Volunteering Information.* LISP-CRITIC [26] is a knowledge-based system that critiques LISP programs. The interaction is controlled by the user, who selects parts of programs and asks the systems for help in improving the code either for human comprehensibility or machine efficiency. Humans often learn by receiving answers to questions that they did not or could not pose. The active help system ACTIVIST [27] volunteers information that was not requested. ACTIVIST "looks over the shoulder" of a user working with an editor, infers the intended goal from user actions, and volunteers editing advice.
- *SYSTEMS' ASSISTANT: Information Volunteering by Users.* Despite the fact that communication capabilities such as *mixed-initiative dialogues* [28–29] have been found to be crucial for cooperative systems, the progress in achieving them has been rather modest. SYSTEMS' ASSISTANT [30] was an effort to support more mixed-initiative dialogues by allowing users to volunteer information. One of the major findings in building SYSTEMS' ASSISTANT was that to provide a more mixed-initiative interaction style requires more elaborate underlying

knowledge structures and not just a change of the interface.

- *FINANZ: Enriching Systems with Domain-Oriented Abstractions.* FINANZ is a knowledge-based spreadsheet system [6]. Rather than forcing financial experts to describe their problems to programmers, who then build spreadsheet models, FINANZ builds higher-level abstractions related to the financial expert's problem domain. This allows the expert to interact with the computer system using abstractions specific to the problem domain, thereby supporting human problem-domain communication [31].

Building HELGON, LISP-CRITIC, ACTIVIST, SYSTEMS' ASSISTANT, and FINANZ deepened our understanding of iterative problem specification, information volunteering, mixed-initiative dialogues, and human problem-domain communication. Although each of these systems explored issues of importance and each made an identifiable step forward, they all fell short in supporting truly cooperative systems. The systems were isolated efforts and were built for relatively simple domains. One of the lessons learned was that cooperative problem-solving systems are very resource intensive. The next section argues why high-functionality computer systems are the foundations upon which these systems must be built.

2.4. *High-Functionality Computer Systems*

Computer systems should be both *usable* and *useful* [11]. For a system to be useful for a broad class of different tasks, it must offer broad functionality. Computing systems have been moving more and more toward high-functionality systems. In our own work, we have analyzed the Symbolics Lisp machine as a high-functionality computer system. To get a feel for how complexity has evolved from simple programming languages, consider a comparison of the Pascal language with the Symbolics Lisp Machine programming environment shown in Figure 3.

The more powerful systems become, the more difficult they are to use. Before users will be able to take advantage of the power of high-functionality computer systems, the cognitive costs of mastering them must be reduced. The following

Pascal Language vs Symbolics Environment		
	Pascal	**Symbolics**
Functions; Operators	29 functions and procs 19 infix operators	627 CL functions 31352 functions total
Classes	None	3322 with 17305 methods
Control Structures	7	38 special forms 45 CL macros
Documentation	300 pages average book	4400 pages in manuals

Fig. 3. Low- vs. high-functionality systems.

problems of high-functionality systems (as identified by Draper [32], Fischer [11], and Lemke [33]) must be overcome:

- *Users do not know about the existence of tools.* Users cannot develop complete mental models of high-functionality systems. Without complete models, users are sometimes unaware of the existence of tools. A passive help system is of no assistance in these situations. Active systems and browsing tools let users explore a system, and critics [34] point out useful information.
- *Users do not know how to access tools.* Knowing that something exists does not necessarily imply that users know how to find it.
- *Users do not know when to use tools.* In many cases, users lack the *applicability conditions* for tools or components. Features of a computer system may have a sensible design rationale from the viewpoint of system engineers, but this rationale is frequently beyond the grasp of users, even those who are familiar with the basic functions of the system. Systems seem imponderable because users have to search through a large list of options and do not know how to choose among them.
- *Users cannot combine, adapt, and modify tools according to their specific needs.* Even after having overcome all of the previous problems (i.e., a tool was found, its functioning was understood, etc.), in many cases the tool does not do exactly what the user wants. This problem requires system support to carry out modification at an operational level with which the user is familiar.

One major issue that is not directly related to high-functionality systems but nevertheless plays an important part in their effectiveness is that users do not have well-formed goals and plans.

Problem-solving in ill-defined domains can be characterized by the fact that no precise goals and specifications can be articulated. Users of high-functionality systems suffer from a lack of knowledge of the interdependencies between problem specification and which tools exist to solve these problems. Unfamiliarity with this mapping leads users to concentrate too quickly on implementation issues, and they often overlook alternative solutions.

3. Success Models for Cooperative Problem Solving

To deepen our understanding of the problems of high-functionality systems and find ways to overcome these problems, we engaged in a search for success models of such systems. The success model idea has proven to be of great value. We have previously analyzed skiing as a success model [4] and derived architectural components for computer-based learning environments [35] from this analysis. In a similar fashion, the ideas behind spreadsheets were used as guiding principles in system-building efforts in other domains [6,36–37]. Studying success models can provide us with equally important insights as studying the role of failures [38] and their impact on the advancement of design.

In this section, we describe a study done at McGuckin Hardware, argue why it is relevant to these issues, and show how the store has successfully addressed the difficult problems mentioned previously. We then place the study in the larger context of research in situated cognition.

3.1. McGuckin: An Empirical Study

A preliminary analysis indicated the McGuckin Hardware in Boulder, Colorado, might be an ideal candidate for a success model. McGuckin carries more than 350,000 different line items in 33,000 square feet of retail space. The store's superior reputation among its customers and its continued growth and profitability make it a success model.

To get a better understanding of just how the "system" operates, we asked McGuckin Hardware for permission to observe and record interactions between customers and sales agents. Some of the dialogues were transcribed from audiotapes and carefully analyzed. Videotapes would have been a superior medium, but would have interfered too much with store operations.

The decision to observe directly as people do problem solving and design in the real world was made as a result of considering the perspective of situated cognition research. Lave [10], Schoen [7], and others have shown how problem solving in daily activity is shaped by the dynamic encounter between the culturally endowed mind and its total context. This leads to a vision of cognition as a dialectic between persons acting and the settings in which their activity is constituted. Lave [10] argued that theoretically charged, unexamined, normative models of thinking lose their descriptive and predictive power when research is moved to everyday settings and relaxes its grip on the structuring of activities.

The following dialogues illustrate the inherent difficulties in high-functionality systems mentioned above (for additional details of our study see Reeves [39]).

Users do not know about the existence of tools. In this dialogue, the customer is unsure about how to attach a sign to a metal pole. The customer does not know of self-tapping bolts and therefore cannot ask for them. Even if we assume a complete understanding of the problem, this is not enough to guarantee the knowledge of the best tool for the problem. Here the customer ends up buying a fastener that is introduced and explained by the salesperson.

Dialogue$_1$: Attaching a Sign to a Square Metal Pole[1]

1. C: *I'm looking for a small fastener maybe one-sixteenth.*
2. S: Okay. Plastic? Metal?
3. C: *Well, what I've got is to fasten a sign on to a square pole. I've got a hole in the top and it fits fine and I got to get one on the bottom.*

   After looking at several fasteners, and asking a few more questions, the salesperson suggests a certain type of fastener.

4. S: How about a self-tapping bolt?
   Picks one up and shows it.
5. C: *Well, what uhh, well, this would probably do it, what about, would it come back out?*
6. S: Oh sure. It'd come back out.
7. C: *But once it's in?*
8. S: As long as the hole is smaller than this thing, you can thread it in and out.

Users don't know how to access tools. The next dialogue shows that it can be difficult just to find items you "know" exist. The customer is specific about the wanted item and even seems to know the store fairly well, but still cannot find the item.

Dialogue$_2$: Finding Tool Clips

1. C: *I need clips for tools where you shove it up in them and it holds*
2. S: Yeah.
3. C: *I mean not just a single clip, a bunch of them. We tried in housewares, the cheap little ones, tools only have like funny kind of ones. Where else could they be?*
4. S: Garden center, for rakes and shovels and things like that
5. C: *Would it be there?*
6. S: Yeah.
7. C: *Okay, I know where that is, thanks.*

Users do not know when to use tools. The interaction shown in Dialogue$_3$ involves a search for scales to weigh small animals and illustrates the concept of *applicability conditions*: the conditions under which an item can be used, especially for "unintended" purposes. The salesperson is able to recognize a crucial element: namely, that there be a platform large enough to hold something of a certain approximate size and weight. He helps the customer to know *when* to use a given tool, even though that use might not have been intended by the designer of the tool. The fact that a scale is intended for food is less important than those features.

Dialogue$_3$ also illustrates the use of *differential descriptions*. The customer describes the intended item "differentially" in terms of an example, building on what the environment has to offer. The customer uses an example item (the "little tiny ones over here") to differentially describe the intended solution. The salesperson extracts the crucial information and suggests an item intended for a different domain, yet useful for accomplishing the described task.

In Dialogue$_4$ the customer wants strength, but the salesperson has to point out a crucial feature of that strength: that it comes at the expense of brittleness.

Dialogue$_3$: Scales for Small Animals

1. C: *I'm looking for some scales and I saw some little tiny ones over here, but I need something that has a large platform on it, to weigh small animals on.*
   Holds hands about 18 inches apart.
2. S: I would think something in our housewares department, for weighing food and things like that. Go on down to the last isle on the left.
3. C: *Okay.*

Dialogue$_4$: Hardened Bolts

1. C: *So if I were going to hook something would this be the best thing? What I'm going to have is I'm going to drill into the cement and have it sticking out.*
2. S: You going to have this sticking out, just the shaft of the bolt? holding a bolt and pointing to the unthreaded shaft.
3. C: *Right.*
4. S: Hmm. Interesting problem.
5. C: *A hardened bolt would give me more . . .*
6. S: Yeah, but it'll shear, they're more brittle. I don't know if you'd be any better off with a hardened.

Users can not combine or adapt tools for special uses. Although the combination in Dialogue$_5$ is simple, it does illustrate how tools can be combined in various ways. The customer doesn't know why the salesperson suggests a certain combination of tools, but ventures a guess. The salesperson allows the suggestion, but then states his reason.

Dialogue$_5$: Combining Simple Tools

1. S: After deciding that a three-sixteenth inch wire is to be looped around a half-inch bolt, which is mounted in cement
   You want a small enough loop, put it between two washers. Picks up two washers and places them on the shaft of a bolt.
2. C: *Small enough loop.*
3. S: Yeah.
4. C: *Why between two washers, so it won't rub?*
5. S: Yeah, so it won't slide off. Probably won't.

Observing interactions like these confirmed the previous analysis of the difficulties of using high-functionality systems. In addition, it raised several other issues that must be considered in building cooperative problem-solving systems.

Incremental problem specifications. Dialogue$_6$ shows that there is a close relationship between defining specifications incrementally, as seen here, and establishing shared knowledge, as will be seen in the next dialogue. The distinction is a subtle, yet useful one. Shared knowledge has more to do with establishing a common reference point with which to discuss a situation, and less to do with the specific process of identifying relevant parts of the problem domain.

Dialogue$_6$: Incrementally Refining a Query

1. C: *I need a cover for a barbecue.*
2. S: (Leading customer down an isle where several grills are lined up and accessories are displayed) Okay . . . what have we got here . . . chaise, chair barbecue grill cover. . . . Does that look kind of like what you got? (Pointing to one of the grills.) Similar? No?
3. C: *No.*
4. S: Take any measurements?
5. C: *No.*
6. S: That's a good guess there. (Pointing to a one-burner grill.)
7. C: *It's a double burner one.*
8. S: 52 inches. That's the total length it'll cover. (Measuring with the tape and holding the tape over one of the grills.)
9. C: *Yeah. I know it's not that big at all.*
10. S: You saying about 18 by 18. Well, this is 27, it'll cover up to here. (Using measuring tape again and pointing.)
11. C: *I need two.*
12. S: A couple . . . in that brand, that's all I have. Here are these Weber ones, thicker material and all that. Here are some smaller ones.
13. C: *I'll take this one.*
14. S: We'll be getting more of these pretty soon.
15. C: *You'll have them by Christmas?*
16. S: Hopefully Thursday.

Achieving shared understanding. Between the time a customer begins to interact with a sales agent and the time the customer leaves with a "satisficing" solution [23], a shared understanding must be created between the two cooperating agents. The customer must begin to appreciate relevant parts of the solution domain and the sales agent must understand the problem in enough depth to make reasonable suggestions. Dialogue$_7$ shows how establishing shared understanding is a gradual process in which each person participates, sometimes ignoring questions, sometimes volunteering information, and sometimes identifying miscommunications. Illustrated also are the problems of knowing about the existence of tools and understanding the results that they produce. The customer wants to fasten a sign to a square metal pole. The top of the sign has been fastened via a preexisting hole, but the bottom is still unattached. The customer learns about certain fasteners while the salesperson learns about the specific problem. Their shared understanding increases as each in turn asks questions and makes suggestions that are critiqued by the other.

Dialogue$_7$: Attaching a Sign to a Square Metal Pole[2]

1. C: *I'm looking for a smaller fastener. Maybe one-sixteenth.*

2. S: Okay. Plastic? Metal?
3. C: *Well, what I've got is to fasten a sign onto a square pole. I've got a hole in the top and it fits fine and I've got to get one on the bottom.*
4. S: Pole have holes in it?
5. C: *Yeah. I had a one-eighth bolt, but it's too big. Need something smaller than that.*
6. S: Round pole? Square Pole?
7. C: *Square pole.*
8. S: (Picking up a fastener and showing it) You tried these?
9. C: (Scrutinizing the fastener.) *Hmmmm.*
10. S: You've got to have a five-sixteenths hole and you fold this thing up and stick it in. Would that work?
11. C: *It's got to be five-sixteenths?*
12. S: Yes. The size of the shaft on this thing. (Pointing to the fastener.)
13. C: *It's not that big.*
14. S: No way to drill it?
15. C: *No.*
16. S: No. What did you use the first time?
17. C: *I tried a one-eighth inch.*
18. S: How thick is the metal in the pole?
19. C: *Oh, probably about one eighth inch.* (Pointing to a certain fastener.) *How about these?*
20. S: (Picking one up and showing the moving parts.) These work on hollow-core doors.
21. C: *Yeah.*
22. S: (Walking over to a different kind of fastener and picking it up) I don't know if this would be strong enough. Still need a three sixteenth hole. If the wind is blowing hard it might give way. Just putting it in with a screwdriver?
23. C: *Yeah.*
24. S: How about a self-tapping bolt? (Picks one up and shows it.) Put that in, tighten it down. (points to tip), that's a thread cutting thread there.
25. C: *Well, what, uhmm, well, this would probably do it. What about, would it come back out?*
26. S: Oh sure. It would come back out.
27. C: *But once it's in?*
28. S: As long as the hole is smaller than this thing, you can thread it in and out.

Integration between problem setting and problem solving. Dialogue$_8$ shows an interaction in which a customer wanted to buy heaters, then decided to reconceptualize the problem from one of "adding heat," to one of "retaining heat." This appears to be a trivial reframing and hardly worth notice, but we will argue that understanding exactly this kind of reframing is crucial to building cooperative problem-solving systems. The problem *itself* was redefined.

Dialogue$_8$: Generating Versus Containing Heat

1. C: *I want to get a couple of heaters for a downstairs hallway.*
2. S: What are you doing? What are you trying to heat?
3. C: *I'm trying to heat a downstairs hallway.*
4. S: How high are the ceilings?
5. C: *Normal, about eight feet.*
6. S: Okay, how about these here? They proceed to agree on two heaters.
7. C: *Well, the reason it gets so cold is that there's a staircase at the end of the hallway*
8. S: Where do the stairs lead?
9. C: *They go up to a landing with a cathedral ceiling.*
10. S: Ok, maybe you can just put a door across the stairs, or put a ceiling fan up to blow the hot air back down.

Summary. The findings of the McGuckin study can be summarized as follows:

Natural Language is less important than Natural Communication. People rarely spoke in complete, grammatical sentences, yet managed to communicate in a natural way. In fact, most of the dialogues shown here had to be "cleaned up" for readability. The study provided convincing evidence that the support for *natural communication* [29], allowing for breakdowns, clarifying dialogues, explanations, etc., is more important for cooperative problem solving than being able to parse syntactically complex sentences. One objective of future human-computer communi-

cation research should therefore be to understand the processes of intention communication and recognition well enough to enable a system to participate in a natural dialogue with its user [40].

Multiple Specification Techniques. Customers used a great variety of specification techniques such as bringing in a broken part, pointing to an item in a catalog or in the store, and giving general descriptions such as "I need a lock that qualifies for cheaper insurance rates."

Mixed Initiative Dialogues. People were flexible in the roles they played during a problem-solving episode. They easily switched from asking to explaining, from learning to teaching. Because $Dialogue_7$ is the longest, it probably shows this best. The structure of these dialogues was determined neither by the customer nor by the sales agent, but clearly indicated mixed initiative [28] determined by the specifics of the joint problem-solving effort.

Management of Trouble. Many breakdowns and misunderstandings occurred during the observed problem-solving episodes, but in almost all cases clarifying dialogues allowed their recovery. Problem solving among humans cannot be characterized by the absence of trouble, but by the identification and repair of breakdowns [8]. $Dialogue_6$ and $Dialogue_7$ contain examples of this.

Simultaneous Exploration of Problem and Solution Spaces. Customers and sales agents worked within both problem and solution spaces simultaneously, or at least alternatively. Typically the problem owner (customer) had a better grasp of the problem space and the problem solver (sales agent) had a better understanding of the solution space, but over time these spaces converged until there was a large enough intersection of shared knowledge within which potential solutions could be evaluated. This is seen in Dialogue- in which the customer knows what needs to be done but needs a better understanding of the possible solutions, and the salesperson knows how many different fasteners work but needs to understand the specific application.

Humans operate within the physical world. Although perhaps obvious, system designers overlook the fact that people use elements of the physical world as sources of information, as reminders, and in general as extensions of their own knowledge and reasoning systems. In most of the dialogues that deal with fasteners, both the customer and salesperson held the items and used them to guide and clarify the discussion. For example, in $Dialogue_5$, the salesperson picked up two washers and placed them on the shaft of the bolt, leaving a small gap between them to show where the cable would go, and how the loop needs to be small enough to be guided or constrained by the washers.

Humans make use of distributed intelligence. Much of people's intelligent behavior results from the interaction of mental processes with the objects and constraints of the world, and much behavior takes place through a cooperative process with others. Collaborators challenge each other's analysis of the problem and help to achieve creative solutions. One thing that surfaced in discussions with salespeople is that when they send a customer to another department, they count on the customer being able to find the items, but also expect another salesperson to be available there.

3.2. A Situated Cognition Perspective

The perspective of situated cognition researchers is important in this analysis of cooperative problem-solving success models. McGuckin Hardware provides an example of what situated cognition researchers have been claiming: much of problem solving is fundamentally related to the larger context in which the problem gets perceived, framed, and eventually resolved. Suchman [8] argued that plans are just one of the resources in the problem-solving process, not *the* guiding principles. The McGuckin study confirms this view of plans: customers do have plans, but these plans are just one resource, not the primary guide.

Lave [10] argued that the problem-solving context plays a crucial role in problem framing. The McGuckin study confirmed this finding. As customers interacted with the wide variety of hardware (e.g. two isles of fasteners), they were able to recast their perception of the problem they came in to solve.

In a critique of the approach technical rationality has encouraged professional practitioners to take toward ill-defined problems. Schoen [7] argued against abstract principles and for skills

developed in domain-specific problem solving, emphasizing the role that problem setting plays. Real problems are never given, but "must be constructed from the materials of problematic situations which are puzzling, troubling, and uncertain."

The setting of a problem is as important as the problem itself. The word "setting" means two things: (1) the physical and social environment (the "context") in which a problem solver acts, and (2) the process of defining the problem. The problem context provides key resources in solving the problem, because it affects how we come to perceive the problem and the resources available to us.

Carraher, Carraher, and Schliemann [41] described how important the setting was to Brazilian school children who worked as street vendors. On the street, they were quite accurate in their calculations (98 percent correct), but when given mathematically identical problems outside the marketplace context, their accuracy dropped to a dismal 37 percent.

Attempts have been made to bring more of the environment into consideration in analyzing problem solving. Larkin et al. [42] studied several issues, such as the interaction between the person solving a problem and external memory aids such as paper and pencil, and representing situations that change over time. Situated cognition appears to push this concept of setting and problems that change over time even further into our physical environment and social relations.

When people encounter problems, these problems are embedded in an environment that provides ways in which the problem is perceived and resources with which to analyze it. Problems can be divorced neither from the social settings in which they occur, nor from the process of problem defining. The former provides structuring resources to the problem solver and the latter affects how the problem is allowed to evolve.

Problems and solutions coevolve—one cannot exist without the other. Empirical studies of people developing complex computer systems [21] have confirmed that often the problem is not to implement a given specification, but rather expressing the problem itself: deciding what problem to solve.

In the context of design problems, Rittel [43] argued that "*you cannot understand the problem without having a concept of the solution in mind; and that you cannot gather information meaningfully unless you have understood the problem but that you cannot understand the problem without information about it.*" Taken literally, this leaves no room for a beginning, but there is a way in which this view nevertheless makes sense. If one cannot begin one without the other, then the only way to proceed is with both simultaneously. In problem solving, people cannot proceed until they have a "resolution shape—a sense of an answer and a process for bringing it together with its parts" [10, p. 19].

John Dewey noted that "discovering a problem is the first step in knowing" (cited in [44]). And Wertheimer [45] observed that: "Often in great discoveries the most important thing is that a certain question is found. Envisaging, putting the productive question is often more important, often a greater achievement than the solution of a set question" (cited in VanGundy [44, p. 102]).

We are trying to understand what this means to designers of cooperative problem-solving systems. Success models of these systems provide a new perspective that informs the design of such systems built on a computing platform. Studying people at McGuckin provided an opportunity to observe "everyday cognition." These observations confirm the importance that the situated cognition perspective brings to the design of cooperative problem-solving systems.

4. Second Generation of Cooperative Problem-Solving Systems

In this section, findings from the McGuckin study are related to the framework suggested in the first section. This is followed by a description of a prototype of an integrated, domain-oriented, knowledge-based design environment.

4.1. Requirements for Cooperative Problem-Solving Systems

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 (such as the DIPMETER advisor [46]), 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 percent* compared to 8 percent for the inference engine and 22 percent for the knowledge base. Similar data are reported for commercial knowledge-based system tools (e.g., in Intellicorp's tools, 55–60 percent of the code is interface related [47]). 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 depend critically on each other:

- the structure of the knowledge and problem-solving system itself—how a system represents 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; for this a system must contain a model of the user
- the presentation of this knowledge on the screen; this part is mostly (explicitly or implicitly) associated with user-interface research.

Problems can be fully articulated only in the context of solving them. The McGuckin study clearly indicated that problems in realistic situations are not fixed targets. The combination of a large selection of objects and knowledgeable sales agents creates an environment in which customers can produce partial solutions and get feedback from the items in the store and from the sales agents in the form of critiques. As problem solvers tentatively explore possible solutions and evaluate how those affect their perception of the original problem, they shape the situation; in accordance with their initial appreciation of it, the situation "talks back," and they respond to the situation's back-talk [7].

The fidelity of the design situations' "back talk" must be increased. Many of the problems that are discussed at McGuckin are ill-defined. The artifacts and inventory at McGuckin are powerful to the extent that the sales agents are knowledgeable. Providing rich functionality without domain-specific expertise is not enough. In our system-building work, we originally believed that domain-oriented construction kits would be powerful enough to "talk back" by themselves, but this turned out not to be the case [31]. Construction kits support the construction of an artifact, but they do not provide any feedback on the quality of the design. Knowledgeable sales agents provide this higher level expertise and so help the situation to "talk back."

McGuckin hires experts in the various departments and considers previous experience within a field, such as plumbing, to be more important than previous sales experience in that field. The difference between working and selling experience in a field is crucial. Behind the surface or syntactic layer of the inventory, there is a semantic understanding of trade-offs, and experience in mapping specific problems to multipurpose tools.

There is a need for specialization and putting knowledge in the world. Simon [23] predicted that when a domain reaches a point where the knowledge for skillful professional practice cannot be acquired in roughly a decade, a burden on mastering all the tools and the knowledge will occur [48]. Simon predicted that the following adaptive developments will occur: (1) specialization will increase and (2) practitioners will make increasing use of books and other external reference aids in their work [49]. McGuckin addresses the tool mastery burden by (1) organizing functionality according to external task domains, and (2) incrementally making the information space relevant to the task at hand by an evolving shared understanding between customers and salespeople.

Supporting human problem-domain communication with domain-oriented architectures. The

McGuckin study illustrates the need to respond to a diverse set of tasks. There is an important need in computer science to develop domain-oriented architectures in order to avoid the pitfall of excess generality. Instead of serving all needs obscurely and insufficiently with general purpose programming languages, domain-oriented architectures serve a few needs well. The semantics of our computing environments need to be better tuned to specific domains of discourse; this involves support for different kinds of primitive entities, for specification of properties other than computational functionality, and for computational models that match the users' own models. Human-computer communication needs to be advanced to human-problem domain communication, where the computer becomes "invisible" and users have the feeling of interacting directly with a problem domain.

4.2. *Integrated, Domain-Oriented, Knowledge-Based Design Environments*

The requirements articulated for the second generation of cooperative problem solving systems lead us to the development of *integrated domain-oriented design environments*. Over the last few years, design environments were developed in the following areas: (1) user interface design [50], (2) kitchen design [20,51], (3) COBOL programming [52], (4) design of decision support system for water management [53], and (5) computer network design [54]. In this section, we will first describe a general architecture for design environments, then illustrate it with a specific example and discuss how information can be made relevant to the task at hand in such environments.

A multifaceted architecture. From the individual design efforts, we have developed the general architecture as shown in Figure 4. This multifaceted architecture consists of the following five components:

- A *construction kit* (see Figure 5) is the principal medium for modeling a design. It provides a palette of domain concepts and supports construction using direct manipulation and electronic forms.
- An *argumentative hypermedia system* (see Figure 6) contains issues, answers, and arguments about the design domain.

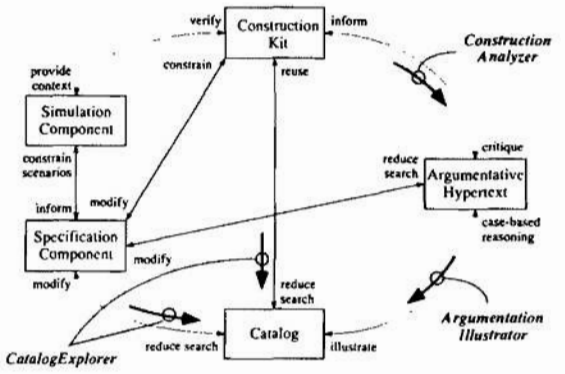

Fig. 4. A multifaceted architecture.

The components of the multifaceted architecture. The links between the components are crucial for exploiting the synergy of the integration.

- A *catalog* (see Figure 5) is a collection of prestored designs that illustrate the space of possible designs in the domain and support reuse and case-based reasoning.
- A *specification component* (see Figures 7 and 8) allows designers to describe characteristics of the design they have in mind. The specifications are expected to be modified and augmented during the design process, rather than to be fully articulated at the beginning. They are used to retrieve design objects from the catalog and to filter information in the hypermedia information space.
- A *simulation component* allows designers to carry out "what-if" games to simulate various usage scenarios involving the artifact being designed.

JANUS: An example. JANUS [20], a design environment to support kitchen designers, will be used as an example to illustrate our approach. JANUS-CONSTRUCTION is the construction kit for the system. The palette of the construction kit contains domain-oriented building blocks called design units, such as sink, stove, and refrigerator (Figure 5). Designers construct by selecting design units from the palette and placing them into the work area. In addition to design by *composition* (using the palette and constructing an artifact from scratch), JANUS-CONSTRUCTION also supports design by *modification*. Ex-

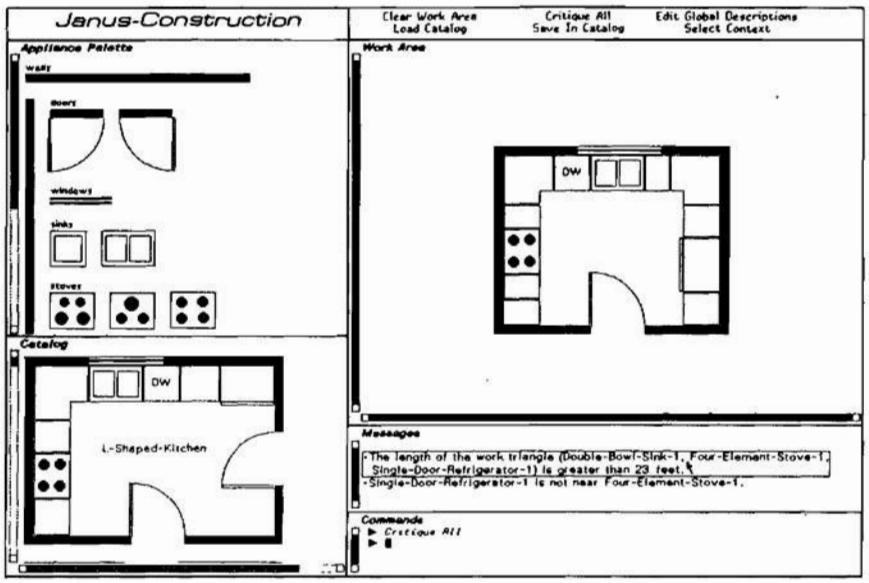

Fig. 5. JANUS-CONSTRUCTION: The work triangle critic.

JANUS-CONSTRUCTION is the construction part of JANUS. Building blocks (design units) are selected from the *Palette* and moved to desired locations inside the *Work Area*. Designers can reuse and redesign complete floor plans from the *Catalog*. The *Messages* pane displays critic messages automatically after each design change that triggers a critique. Clicking with the mouse on a message activates JANUS-ARGUMENTATION and displays the argumentation related to that message (see Figure 6).

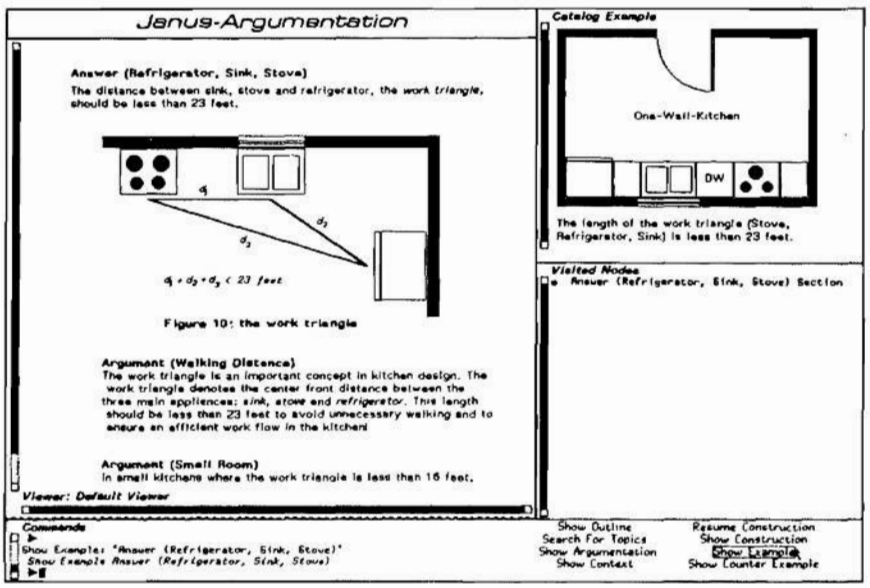

Fig. 6. JANUS-ARGUMENTATION: Rationale for the work triangle rule.

Fig. Janus-Argumentation is an argumentative hypermedia system. The Viewer pane shows a diagram illustrating the work triangle concept and arguments for and against a work triangle answer. The top right pane shows an example illustrating this answer generated by the ARGUMENTATION-ILLUSTRATOR. The *Visited Nodes* pane lists in sequential order the previously visited argumentation topics. By clicking with the mouse on one of these items, or on any bold or italicized item in the argumentation text itself, the user can navigate to related issues, answers, and arguments.

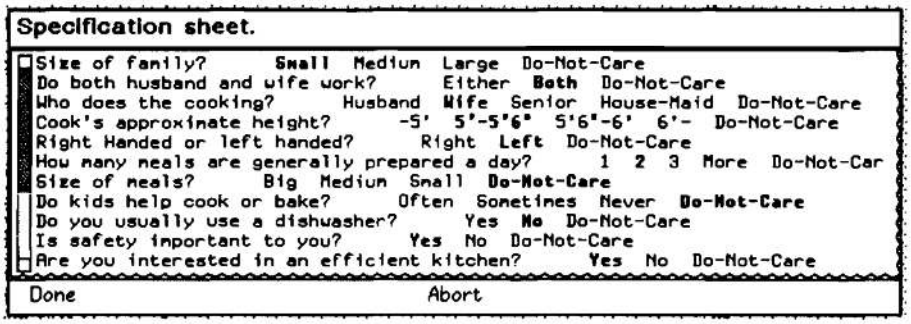

Fig. 7. Specification sheet.

The *Specify* command in CATALOG-EXPLORER provides a specification sheet in the form of a questionnaire.

isting designs can be modified by retrieving them from the catalog and manipulating them in the work area.

Designers using JANUS-CONSTRUCTION experienced a sense of accomplishment in using the system because it enabled them to construct something quickly without having detailed knowledge about computers. But construction kits do not in themselves lead to the production of flawless artifacts [31] because they do not help designers discover the shortcomings of the artifact they are constructing. As passive representations, constructions in the work area do not talk back unless designers have the skill and experience to form new appreciations and understandings when constructing. Designers often do not see characteristics that lead to breakdowns in real use situations.

Critics. Critics [34] operationalize the concept of a situation that "talks back" [7]. They use knowledge of design principles to detect and critique partial and suboptimal solutions constructed by the designer. The critics in JANUS-CONSTRUCTION identify potential problems in the artifact being designed. Their knowledge about kitchen design includes design principles based on building codes, safety standards, and functional preferences. Critics are implemented as condition-action rules, which are tested whenever the design is changed. The changes that trigger a critic are operations that modify the design in the work area. When a design principle is violated, a critic will fire and display a critique in the messages pane of Figure 5. In the figure, the work triangle critic fired telling the designer that the "work triangle is greater than 23 feet." This identifies a possibly problematic situation (a breakdown), and prompts the designer to reflect on it.

Lack of Argumentative Support. The advan-

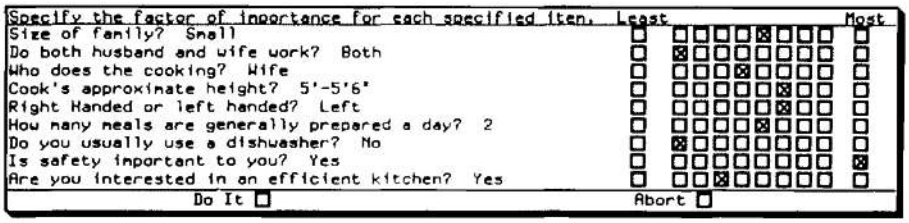

Fig. 8. Weighting sheet for the specification.

After specification, users can weigh the importance of each specified item.

tage of constructing something is that the constructed artifacts and situations can "talk back" to the designer. But the short messages the critics present to designers cannot reflect the complex reasoning behind the corresponding design issues. To overcome this shortcoming, we initially developed a static explanation component for the critic messages [50]. The design of this component was based on the assumption that there is a "right" answer to a problem. But the explanation component could not support the deliberative nature of design problems. Therefore, argumentation about issues raised by critics must be supported, and argumentation must be integrated into the context of construction.

JANUS-ARGUMENTATION. JANUS-ARGUMENTATION is the argumentation component of JANUS (Figure 6). It is an argumentative hypermedia system implemented using the SYMBOLICS DOCUMENT EXAMINER [55]. JANUS-ARGUMENTATION offers a domain-oriented, generic issue base about how to construct residential kitchens. With JANUS-ARGUMENTATION, designers explore issues, answers, and arguments by navigating through the issue base. The starting point for the navigation is the argumentative context triggered by a critic message in JANUS-CONSTRUCTION. Clicking with the mouse on a critique in JANUS-CONSTRUCTION (Figure 5) activates JANUS-ARGUMENTATION and accesses issues and answers corresponding to the critique.

Domain orientation. The substrate used to design computer-based artifacts typically consists of low-level abstractions (such as statements and data structures in programming languages, and primitive geometric objects in engineering computer-aided design). Abstractions at that level are far removed from the concepts that form the basis of thinking in the application domains in which these artifacts are to operate. Our design environments support *human problem-domain communication* [31] by allowing designers to build artifacts from application-oriented building blocks according to the principles of that domain—not the principles of software or geometry.

Integration. The multifaceted architecture derives its essential value from the integration of its components and links between the components. Used individually, the components are unable to achieve their full potential. Used in combination, each component augments the value of the others, forming a synergistic whole. At each stage in the design process, the partial design embedded in the design environment serves as a stimulus to users, suggesting what they should attend to next. Links among the components of the architecture are supported by various mechanisms (see Figure 4):

- *CONSTRUCTION-ANALYZER.* The CONSTRUCTION-ANALYZER is a critiquing system that provides access to relevant information in the argumentative issue base. The firing of a critic signals a breakdown to users and provides them with an entry into the exact place in the argumentative hypermedia system where the corresponding argumentation is located.
- *ARGUMENTATION-ILLUSTRATOR.* The explanation given in argumentation is often highly abstract and very conceptual. Concrete design examples that match the explanation help users to understand the concept. The ARGUMENTATION-ILLUSTRATOR (see Figure 6) helps users to understand the information given in the argumentative hypermedia by finding a catalog example that illustrates the concept.
- *CATALOG-EXPLORER.* CATALOG-EXPLORER helps users to search the catalog space according to the task at hand [56]. It retrieves design examples similar to the current construction situation, and orders a set of examples by their appropriateness to the current specification.

Next we describe in more detail the system components that link the specification, construction and the existing information spaces. We focus on this part of our overall system-building effort, because it illustrates most clearly some of the lessons we have learned in the McGuckin study.

Making information relevant to the task at hand. To integrate problem setting and problem solving in design environments, it is crucial to support retrieval of information relevant to the task at hand. Every step made by a designer toward a solution determines a new space of related information, which cannot be determined a priori. Conventional information retrieval techniques are thus not applicable for design environments [17]. In a conventional *query-based*

search, a specific query has to be formulated. Once users can articulate what they need, a query-based search takes away much of the burden of locating promising objects [57]. In *navigational access* provided by browsing mechanisms, users tend to get lost looking for some target information if the browsing space is large and the structure is complex [58].

Design environments need additional mechanisms that can identify small sets of objects relevant to the task at hand. The systems must allow users to incrementally articulate the task at hand. The information provided in response to these problem-solving activities must assist users in refining the definition of their problem. A typical cycle of events supported by the multifaceted architecture is: (1) users create a partial specification or partial construction, (2) a breakdown occurs, (3) users switch and consult other components in the system made relevant by the system to the partially articulated task at hand, and (4) users refine their understanding based on "the back talk of the situation". As users go back and forth among these components, the problem space is narrowed, a shared understanding between users and the system evolves, and the artifact is incrementally refined.

CATALOG-EXPLORER. CATALOG-EXPLORER (for details see [56,59]) links the specification and construction components with the catalog in JANUS (see Figure 4). CATALOG-EXPLORER (1) exploits the information articulated in a partial specification to prioritize the designs stored in the catalog, and (2) analyzes the current construction and retrieves *similar* examples from the catalog using similarity metrics. Each design object stored in the catalog of JANUS consists of a floor layout and a set of slot values filled by users. Those design objects can be reused for case-based reasoning such as providing a solution to a new problem, evaluating and justifying decisions behind the partial specification or construction, and informing designers of possible failures [60,61].

CATALOG-EXPLORER extends HELGON and other information retrieval systems by relieving users of the task of forming queries and navigating in information spaces. Due to its integration based on the multifaceted architecture, CATALOG-EXPLORER can capture a user's task at hand by analyzing the partial specification and construction. The system then infers the relevance of stored information to that task.

Retrieval from Specification. As a specification component in the multifaceted architecture, CATALOG-EXPLORER provides (1) a *Specification Sheet* for specifying requirements for a design (see Figure 7), and (2) a *Weighting Sheet* for assigning a weight to each specification item to differentiate the factor of importance (see Figure 8). By analyzing information obtained by those mechanisms, the system reorders catalog examples by computing the *appropriateness* value of each design example according to the given set of weighted specifications.

To capture the user's task at hand from a specification and make design objects relevant to that task by inferring the relevance, one must deal with hidden features, partial matching, and contradictory features of design. To address these issues, the system has *specification-linking rules* for matching between a specification and design objects, and a metric to measure the *appropriateness* of an existing design with respect to a specification.

Specification-linking Rules. There are two types of specification items: *surface features* such as *"a kitchen that has a dishwasher"* and *hidden features* such as *"good for a small family."* Retrieving design examples from the catalog by surface feature specification can be done in a straightforward manner using conventional searching mechanisms. In contrast, retrieval using hidden features requires domain knowledge to infer those features because it is often difficult to determine a priori the features that become important for later recall.

The *specification-linking rules* of CATALOG-EXPLORER link each hidden feature specification item to a set of condition rules. In the integrated environment this domain knowledge can be derived from the contents of the argumentative hypermedia component. Figure 9 illustrates how specification-linking rules can (1) bridge the gap between problem spaces and solution spaces, (2) create a shared understanding by narrowing the information space through extraction of examples from the catalog that are relevant to the task at hand, and (3) infer that a kitchen that has a stove away from both a door

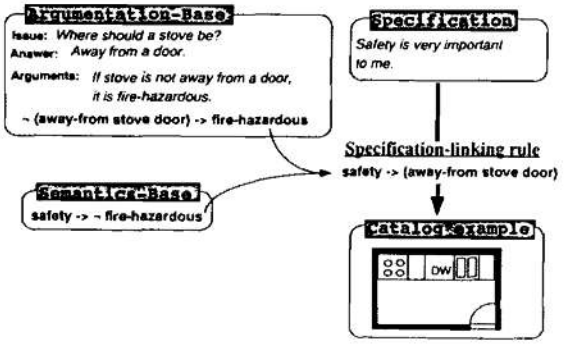

Fig. 9. Specification-linking rules in CATALOG-EXPLORER.

and a window satisfies a hidden feature such as *a safe kitchen*.

4.3. Future Work

Our efforts to develop cooperative problem-solving systems based on the exploration and analysis of a success model opens up a number of research issues for future work.

Transferring success models. Any use of a success model in one domain cannot be transferred to another domain without great care. The most obvious difference in generalizing from our McGuckin study to the development of computer systems is that human-human interaction is *not* human-computer interaction. A major shortcoming of human-computer interaction when compared to human-human interaction is the limited bandwidth of the explicit (facial expressions, gestures) as well as the implicit (shared understanding, mutual intelligibility) communication channel between the cooperating agents [8]. Cooperative systems must have better access to the users' actions and intentions, make clear their own limits, and try to compensate for those limits. The integration of the different components in our design environments is a first attempt to allow users to communicate their goals to the system in the form of partial specifications and partial constructions.

Software environments have different *qualities* and *shortcomings* than a hardware store. A software-specific challenge and opportunity is that of making software truly "soft" by supporting end-user modifiability [62]. Because situations of practice are complex, unique, uncertain, conflicting, and unstable, supporting end-user modifiability is a necessity rather than a luxury for the development of future computer systems. The need to enhance existing systems cannot be restricted to the interface alone, but extends to the system as a whole.

The critical role of examples. Examples play a major role in shaping the problem. This is a difficult issue to study because one cannot know beforehand how examples will affect the problem, and a post-hoc analysis will be blind to the subtle changes in the problem definition that examples have induced. As much as sales agents use analogies and examples to narrow the search down, it would be desirable to get a better grasp of what is involved in creatively generating partial solutions that best clarify the problem itself.

Avoid delegation—put owners of problems in charge. Related to the assumption that problems can be clearly defined is the notion that they can be delegated. If a problem description could indeed grow apart from its solution, then it would be possible to "delegate" that problem description to an intermediary. Compared to problem owners, however, intermediaries have severe limitations acting in an ill-defined problem. A key attribute of a problem is that the owner has the authority to change its description. The difficulty with delegating ill-defined problems is that the owner of the problem interacts only indirectly with the emergent solution and is not able to foresee implications that certain specifications and assumptions are having on the final solution. The way this manifests itself at McGuckin is that sometimes problem owners go to the store themselves whereas at other times they send someone else. Studying these cases may provide us with more insights into constructing computing systems with which problem owners can interact directly. Based on the feedback of the situation and intermediate results of possible solutions, the owner of a problem reinterprets the problem description itself. However, when a problem is delegated this feedback loop is broken.

The McGuckin study suggests that the difficulties of delegating a moving target, i.e., the problem still being defined, should be addressed

by supporting the problem owner rather than building better and better systems for delegatees who have trouble intepreting the problem as they try to solve it. We must build systems that model the intermediate agents between the problem owner and the hardware. Each of the intermediate agents provides expertise, and this must be captured in the interaction with the owner of the problem. Providing a nice interface will not be enough unless we believe that that is all that the intermediate agents do.

Collaborating designers. Support is needed for multiple designers working on the same design artifact. The evolution of the design should be captured as well as the annotations that document the design discussions that took place. One should help users collaborate by more tightly integrating the catalog and argumentative hypermedia components. Rather than carrying out design discussions in the abstract, an annotation component should allow arguments to be made using the design artifact itself. Annotation supports designers arguing about specific design problems and is a step toward capturing design rationale [54].

5. Conclusions

Interfaces (whether intelligent or not) by themselves are not sufficient to make systems more useful and more usable. This position was also articulated by Papert [63]: "*I think the interface is part of a larger thing. I think that putting the emphasis on the interface somewhat confuses the issues. But if only the interface is changed, and what lies behind it and what you can do with the system isn't changed, you're only scratching the surface. The interface is only the surface. I think we need deeper ways to think about differences in computing.*"

Comparing current computer systems to what we saw at McGuckin led us the following claim: "*High-functionality computer systems offer the same broad functionality as large hardware stores, but they are operated like discount department stores*"—what is missing from them is the cooperative support of knowledgeable sales agents. Our efforts to develop conceptual frameworks and prototypes of cooperative problem-solving systems are based on these insights and represent an effort to take us beyond the limitations of interface research as well as the limitations of autonomous expert systems.

Acknowledgments

We thank Scott Henninger, Tammy Sumner, and the anonymous reviewers for helpful suggestions. We thank McGuckin Hardware in Boulder (especially Robb Hight, Randy Dilkes, and Larry Kemmer) for allowing us to record and analyze interactions between customers and sales agents. We thank the members of the Human-Computer Communication research group at the University of Colorado, Boulder, who helped us to develop the ideas and the systems described in this paper. The research was partially supported by grants No. CDA-8420944, IRI-8722792, and IRI-9015441 from the National Science Foundation; grant No. MDA903-86-C0143 from the Army Research Institute; and grants from the Intelligent Systems Group at NYNEX, Software Research Associates (SRA) in Tokyo, and the Colorado Institute of Artificial Intelligence.

Notes

1. In this and the following dialogues, *C: means customer* and S: means sales agent. Our explanations and comments are in this typeface. This dialogue is part of a longer one, Dialogue$_2$.
2. The beginning of this dialogue was also shown in Dialogue$_1$.

References

1. D.C. Engelbart, W.K. English, "A research center for augmenting human intellect," in *Proceedings of the AFIPS Fall Joint Computer Conference*, The Thompson Book Company, Washington, D.C., 1968, pp. 395–410.
2. M.J. Stefik, "The next knowledge medium," *AI Magazine*, vol. 7, pp. 34–46, 1986.
3. G. Fischer, "Communications requirements for cooperative problem solving systems," *The International Journal of Information Systems (Special Issue on Knowledge Engineering)*, vol. 15, pp. 21–36, 1990.
4. R.R. Burton, J.S. Brown, G. Fischer, "Analysis of

skiing as a success model of instruction: Manipulating the learning environment to enhance skill acquisition," in *Everyday Cognition: Its Development in Social Context*, edited by B. Rogoff, J. Lave, Harvard University Press: Cambridge, MA—London, pp. 139–150, 1984.

5. T.W. Malone, "How do people organize their desks? Implications for the design of office information systems," *ACM Transactions on Office Information Systems*, vol. 1, pp. 99–112, 1983.
6. G. Fischer, C. Rathke, "Knowledge-based spreadsheet systems," in *Proceedings of AAAI-88, Seventh National Conference on Artificial Intelligence (St. Paul, (MN)*, Morgan Kaufmann Publishers, San Mateo, CA, 1988, pp. 802–807.
7. D.A. Schoen, *The Reflective Practitioner: How Professionals Think in Action*, Basic Books: New York, 1983.
8. L.A. Suchman, *Plans and Situated Actions*, Cambridge University Press: New York, 1987.
9. T. Winograd, F. Flores, *Understanding Computers and Cognition: A New Foundation for Design*, Ablex Publishing Corporation: Norwood, NJ, 1986.
10. J. Lave, *Cognition in Practice*, Cambridge University Press: Cambridge, UK, 1988.
11. G. Fischer, "Making computers more useful and more usable," in *Proceedings of the 2nd International Conference on Human-Computer Interaction (Honolulu, Hawaii)*, Elsevier Science Publishers, New York, 1987, pp. 97–104.
12. S.K. Card, "Human factors and the intelligent interface," in *Combining Human and Artificial Intelligence: A New Frontier for Human Factors, Symposium for the Metropolitan Chapter of the Human Factors Society*, New York, 1984.
13. R.R. Burton, J.S. Brown, "An investigation of computer coaching for informal learning activities," in *Intelligent Tutoring Systems*, edited by D.H. Sleeman, J.S. Brown, Academic Press: London—New York, chapter 4, pp. 79–98, 1982.
14. B.G. Buchanan, E.H. Shortliffe, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, Addison-Wesley Publishing Company: Reading, MA, 1984.
15. B.G. Buchanan, E.H. Shortliffe, "Human engineering of medical expert systems," in *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, Addison-Wesley Publishing Company: Reading, MA, pp. 599–612, chapter 32, 1984.
16. J. McDermott, "R1: A rule-based configurer of computer systems," *Artificial Intelligence*, vol. 19, pp. 39–88, 1982.
17. G. Fischer, S.R. Henninger, D.F. Redmiles, "Intertwining query construction and relevance evaluation," in *Human Factors in Computing Systems, CHI'91 Conference Proceedings (New Orleans, LA)*, ACM, pp. 55–62, 1991.
18. D.D. Woods, "Cognitive technologies: The design of joint human-machine cognitive systems," *AI Magazine*, vol. 6, pp. 86–92, 1986.
19. T.W. Malone, K.R. Grant, K.-Y. Lai, R. Rao, D. Rosenblitt, "Object lens: A "spreadsheet" for cooperative work," in *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'88)*, ACM, New York, 1988, pp. 115–124.
20. G. Fischer, R. McCall, A. Morch, "JANUS: Integrating hypertext with a knowledge-based design environment," in *Proceedings of Hypertext '89 (Pittsburgh, PA)*, ACM, New York, 1989, pp. 105–117.
21. B. Curtis, H. Krasner, N. Iscoe, "A field study of the software design process for large systems," *Communications of the ACM*, vol. 31, pp. 1268–1287, 1988.
22. R.L. Teach, E.H. Shortliffe, "An Analysis of Physicians' Attitudes," in *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, Addison-Wesley Publishing Company: Reading, MA, pp. 635–652, Chapter 34, 1984.
23. H.A. Simon, *The Sciences of the Artificial*, The MIT Press: Cambridge, MA, 1981.
24. G. Fischer, H. Nieper-Lemke, "HELGON: Extending the retrieval by reformulation paradigm," in *Human Factors in Computing Systems, CHI'89 Conference Proceedings (Austin, TX)*, ACM: New York, pp. 357–362, 1989.
25. M.D. Williams, "What Makes RABBIT Run?" *International Journal of Man-Machine Studies*, vol. 21, pp. 333–352, 1984.
26. G. Fischer, "A critic for LISP," in *Proceedings of the 10th International Joint Conference on Artificial Intelligence (Milan, Italy)*, edited by J. McDermott, Morgan Kaufmann Publishers, Los Altos, CA, August, 1987, pp. 177–184.
27. G. Fischer, A.C. Lemke, T. Schwab, "Knowledge-based help systems," in *Human Factors in Computing Systems, CHI'85 Conference Proceedings (San Francisco, CA)*, ACM, New York, pp. 161–167, 1985.
28. J.R. Carbonell, "Mixed-initiative man-computer instructional dialogues," Report 1971, BBN, 1970.
29. D.G. Bobrow, R.M. Kaplan, M. Kay, D.A. Norman, H. Thompson, T. Winograd, "GUS, A frame-driven dialog system," *Artificial Intelligence*, vol. 8, pp. 155–173, 1977.
30. G. Fischer, C. Stevens, "Volunteering information—Enhancing the communication capabilities of knowledge-based systems," in *Proceedings of INTERACT'87, 2nd IFIP Conference on Human-Computer Interaction (Stuttgart, FRG)*, edited by H.-J. Bullinger, B. Shackel, North-Holland: Amsterdam, pages 965–971, 1987.
31. G. Fischer, A.C. Lemke, "Construction kits and design environments: Steps toward human problem-domain communication," *Human-Computer Interaction*, vol. 3, pp. 179–222, 1988.
32. S.W. Draper, "The nature of expertise in UNIX," in *Proceedings of INTERACT'84, IFIP Conference on Human-Computer Interaction*, Elsevier Science Publishers, Amsterdam, 1984, pp. 182–186.
33. A.C. Lemke, "Design environments for high-functionality computer systems," PhD thesis, Department of Computer Science, University of Colorado, 1989.

34. G. Fischer, A.C. Lemke, R. McCall, A. Morch, "Making argumentation serve design," *Human Computer Interaction*, vol. 6, nos. 3–4, pp. 393–419, 1991.
35. E. Wenger, *Artificial Intelligence and Tutoring Systems*, Morgan Kaufmann Publishers: Los Altos, CA, 1987.
36. K.-Y. Lai, T.W. Malone, "Object lens: A "spreadsheet" for cooperative work," in *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'88)*, ACM, New York, 1988, pp. 115–124.
37. N. Wilde, C.H. Lewis, "Spreadsheet-based interactive graphics: From prototype to tool," in *Human Factors in Computing Systems, CHI'90 Conference Proceedings (Seattle, WA)*, ACM: New York, 1990, pp. 153–159.
38. H. Petroski, *To Engineer Is Human*, St. Martin's Press: New York, 1985.
39. B.N. Reeves, "Locating the right object in a large hardware store—An empirical study of cooperative problem solving among humans," Technical Report CU-CS-523-91, Department of Computer Science, University of Colorado, Boulder, CO, 1991.
40. T. Winograd, "A Language/Action Perspective on the Design of Cooperative Work," *Human-Computer Interaction*, vol. 3, pp. 3–30, 1988.
41. T.N. Carraher, D.W. Carraher, A.D. Schliemann, "Mathematics in the streets and in the schools," *British Journal of Developmental Psychology*, vol. 3, pp. 21–29, 1985.
42. J. Larkin, et al, "Expert and novice performance in solving physics problems," *Science*, vol. 208, pp. 1335–1342, 1980.
43. H.W.J. Rittel, "Second-generation design methods," in *Developments in Design Methodology*, John Wiley & Sons: New York, 1984, pages 317–327.
44. A.B. VanGundy, *Stalking the Wild Solution: A Problem-Finding Approach to Creative Problem Solving*, Bearly Limited: Buffalo, NY, 1986.
45. M. Wertheimer, *Productive Thinking*, Harper & Row: New York, 1959.
46. R.G. Smith, "On the development of commercial expert systems," *AI Magazine*, vol. 5, pp. 61–73.
47. M. Stelzner, M.D. Williams, "The evolution of interface requirements for expert systems," in *Expert Systems: The User Interface*, Ablex Publishing Corporation: Norwood, NJ, pp. 285–306, Chapter 12, 1988.
48. F.P. Brooks Jr., "No silver bullet, essence and accidents of software engineering," *IEEE Computer*, vol. 20, pp. 10–19, 1987.
49. D.A. Norman, *The Psychology of Everyday Things*, Basic Books, New York, 1988.
50. A.C. Lemke, G. Fischer, "A cooperative problem solving system for user interface design," in *Proceedings of AAAI-90, Eighth National Conference on Artificial Intelligence*, AAAI Press/The MIT Press, Cambridge, MA, 1990, pp. 479–484.
51. G. Fischer, A.C. Lemke, R. McCall, "Towards a system architecture supporting contextualized learning," in *Proceedings of AAAI-90, Ninth National Conference on Artificial Intelligence*, AAAI Press/The MIT Press, Cambridge, MA, 1990, pp. 420–425.
52. M.E. Atwood, B. Burns, W.D. Gray, A.I. Morch, E.R. Radlinski, A. Turner, "The grace integrated learning environment—A progress report," in *Proceedings of the Fourth International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems (IEA/AIE 91)*, 1991, pp. 741–745.
53. A.C. Lemke, S. Gance, "End-user modifiability in a water management application," Technical Report, Department of Computer Science, University of Colorado, 1990.
54. G. Fischer, J. Grudin, A.C. Lemke, R. McCall, J. Ostwald, B. Reeves, F. Shipman, "Supporting indirect, collaborative design with integrated knowledge-based design environments," *Human Computer Interaction, Special Issue on Computer Supported Cooperative Work*, vol. 7, no. 3, 1992 (forthcoming).
55. J.H. Walker, "Document examiner: Delivery interface for hypertext documents," in *Hypertext'87 Papers*, University of North Carolina: Chapel Hill, NC, pp. 307–323, 1987.
56. G. Fischer, K. Nakakoji, "Making design objects relevant to the task at hand," in *Proceedings of AAAI-91, Ninth National Conference on Artificial Intelligence*, AAAI Press/The MIT Press, Cambridge, MA, 1991, pp. 67–73.
57. S. Henninger, "Defining the roles of humans and computers in cooperative problem-solving systems for information retrieval," in *Working Notes of the AAAI Spring Symposium Workshop on Knowledge-Based Human Computer Communication*, AAAI, Menlo Park, CA, 1990, pp. 46–51.
58. F.G. Halasz, "Reflections on notecards: Seven issues for the next generation of hypermedia systems," *Communications of the ACM*, vol. 31, pp. 836–852, 1988.
59. G. Fischer, K. Nakakoji, "Empowering designers with integrated design environments," *Artificial Intelligence in Design '91*, Butterworth-Heinemann Ltd: Oxford, England, 1991, pages 191–209.
60. S. Slade, "Case-based reasoning: A research paradigm," *AI Magazine*, vol. 12, pp. 42–55, 1991.
61. J.L. Kolodner, "What is case-based reasoning?" in AAAI'90 tutorial on case-based reasoning, 1990, pp. 1–32.
62. G. Fischer, A. Girgensohn, "End-user modifiability in design environments," in *Human Factors in Computing Systems, CHI'90 Conference Proceedings (Seattle, WA)*, ACM, New York, pp. 183–191, 1990.
63. S. Papert, "Welcome to the BYTE summit: Future directions," *BYTE*, vol. 15, pp. 226–230, 1990.

Gerhard Fischer is a Professor in the Computer Science Department and a Member of the Institute of Cognitive Science at the University of Colorado, Boulder. His research interests include artificial intelligence, human-computer communication, cognitive science and software design. His research has led to the development of new conceptual frameworks and to the design and implementation of a number of innovative systems in the areas of cooperative problem solving, integrated domain-oriented design environments, intelligent support systems and end-user modifiability (including reuse and redesign).

Brent Reeves is a Research Assistant in the Computer Science Department and a member of the Institute of Cognitive Science at the University of Colorado, Boulder. He is interested in artificial intelligence, human-computer interaction, and computer-supported cooperative work.