University of Colorado at Boulder

Kumiyo Nakakoji and Gerhard Fischer
Department of Computer Science

ECOT 7-7 Engineering Center
Campus Box 430
Boulder, Colorado 80309-0430
(303) 492-7514, FAX: (303) 492-2844

Intertwining Knowledge Delivery and Elicitation: A Process Model for Human-Computer Collaboration in Design

Kumiyo Nakakoji[1,2] and Gerhard Fischer[2]

[1]Software Engineering Laboratory
Software Research Associates Inc.,
1-1-1 Hirakawa-cho, Chiyoda-ku, Tokyo 102, Japan

[2]Department of Computer Science and Institute of Cognitive Science
University of Colorado
Boulder, Colorado 80309-0430, USA

Tel: +1 (303) 492-3912 Fax: +1 (303) 492-2844

kumiyo@cs.colorado.edu, gerhard@cs.colorado.edu

APPEARED IN
"KNOWLEDGE-BASED SYSTEMS", SPECIAL ISSUE: HUMAN-COMPUTER COLLABORATION,
BUTTERWORTH-HEINEMANN, OXFORD, UK, 1995

Abstract: Collaboration among designers can be described with an "action-reflection-critique" model in which the explicit representation of the design contributes to a shared understanding and to the articulation of design knowledge. We describe how domain-oriented design environments based on this model support human-computer collaboration by intertwining knowledge *delivery* and *elicitation*. The KID (Knowing-In-Design) system has a shared understanding about the designers' "task at hand" through a partial design requirement specification and a solution. KID delivers design knowledge relevant to this task at hand, and the delivery helps designers uncover tacit design concerns. Designers are encouraged to store the elicited design knowledge in KID, which results in the evolution of the system's knowledge-bases. The evolution affects the system's subsequent behavior by tuning the delivery toward the designers. This cycle of knowledge delivery and elicitation processes supported by KID allows designers to gradually coevolve design requirements and solutions.

Acknowledgements: We thank the HCC group at the University of Colorado, who contributed to the conceptual framework and the systems discussed in this paper. We also thank Barbara Gibbons of Kitchen Connection at Boulder, Colorado, for her valuable time and her comments on our work. We thank Loren Terveen of AT&T Bell Labs, for his comments and suggestions on an earlier version of the paper. The research was supported by the National Science Foundation under grants No. IRI- 9015441 and MDR-9253425; Software Research Associates, Inc. (Tokyo); the Colorado Advanced Software Institute; US WEST Advanced Technologies; and NYNEX Science and Technology Center.

Intertwining Knowledge Delivery and Elicitation: A Process Model for Human-Computer Collaboration in Design

Kumiyo Nakakoji[1,2] and Gerhard Fischer[2]

[1]Practical Software Engineering Laboratory
SRA Inc.,
1-1-1 Hirakawa-cho, Chiyoda-ku, Tokyo 102, Japan

[2]Department of Computer Science and Institute of Cognitive Science
University of Colorado
Boulder Colorado 80309-0430, USA

Tel: +1 (303) 492-3912 Fax: +1 (303) 492-2844

kumiyo@cs.colorado.edu, gerhard@cs.colorado.edu

Abstract. Collaboration among designers can be described with an "action-reflection-critique" model in which the explicit representation of the design contributes to a shared understanding and to the articulation of design knowledge. We describe how domain-oriented design environments based on this model support human-computer collaboration by intertwining knowledge *delivery* and *elicitation*. The KID (Knowing-In-Design) system has a shared understanding about the designers' "task at hand" through a partial design requirement specification and a solution. KID delivers design knowledge relevant to this task at hand, and the delivery helps designers uncover tacit design concerns. Designers are encouraged to store the elicited design knowledge in KID, which results in the evolution of the system's knowledge-bases. The evolution affects the system's subsequent behavior by tuning the delivery toward the designers. This cycle of knowledge delivery and elicitation processes supported by KID allows designers to gradually coevolve design requirements and solutions.

1. Introduction

Design tasks are *ill-defined* (Simon, 1981) and *open-ended* (Rittel, Webber, 1984). Much of the relevant knowledge required for design is tacit (Polanyi, 1966). Design activities are best supported by taking a human-computer collaborative problem-solving approach. In this paper, we describe a model, an architecture, and a prototype system that support designers in collaborating with one another and with a computer system. *Integrated, domain-oriented, knowledge-based design environments* (Fischer, Nakakoji, 1992) augment skills of designers instead of generating solutions for designers, as typified by the design automation approach (Gero, 1989). Design environments are computer systems that provide design media and tools with which designers can represent their design, and intelligent agents that support designers in using the systems' design knowledge for understanding and reflecting on their partial design.

Based on theories of design and models of human-human collaborative problem solving, we have developed the *action-reflection-critique* model for understanding collaboration among stakeholders.[1] Applying this model to human-computer collaborative design has led us to develop an architecture for a design environment that intertwines knowledge *delivery* and *elicitation*. Knowledge delivery is the presentation of information by a design environment relevant to the partially constructed design represented in the environment. The explicit representation of the partial design and the delivered design knowledge serve as a knowledge elicitation mechanism: they may invoke relevant design knowledge of which designers had been previously unaware. Designers may then explicitly represent the elicited knowledge and store it in the system.

In what follows, we illustrate our approach with the KID (Knowing-In-Design) (Nakakoji, 1993) kitchen design environment and demonstrate how the system intertwines knowledge delivery and elicitation in support of designers in coping with design tasks.

2. A Model for Collaborative Design

Collaboration in design requires not just coordinating divided design tasks; it is also important that designers learn and refine their performance in the course of solving a problem based on the evolving shared understanding.

---

[1]Stakeholders of a design task include people with various roles, such as designers, clients, and end-users. To focus our discussion on issues of collaboration between people and computers rather than among different people, we use the term "designers" to refer to stakeholders in general, and do not distinguish among the different roles of stakeholders.

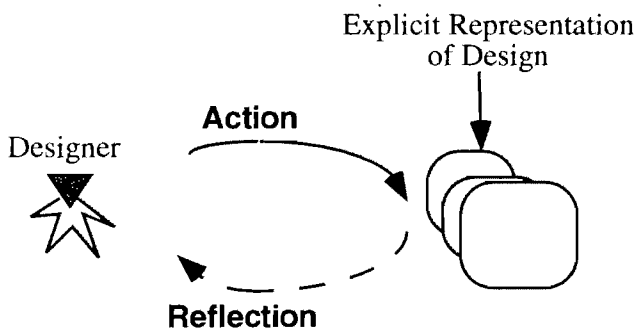

Figure 1: An Action-Reflection Cycle in Design

An action results in the creation and modification of an external design representation, whereas reflection occurs in a designer's mind and may not be explicitly represented. External design representations (such as paper, mockups, or computational artifacts) are essential to support designers to recognize unanticipated consequences of their actions and to reflect upon subsequent actions.

In order to model collaboration in design and further apply it to human-computer collaborative design, we have integrated design theory with models of human-human collaborative problem solving.

2.1. The Action-Reflection Model of Design

According to Schoen's theory, designers work in an alternating cycle of action and reflection (Schoen, 1983). The designer *acts* to shape the design situation by creating or modifying design representations, and the situation "talks back" to the designer, revealing unanticipated consequences of the design actions. In order to understand the situation's back-talk, the designer *reflects* on the actions and consequences, and plans the next course of action (see Figure 1).

2.2. An Action-Reflection-Critique Model for Collaborative Design

In most problem solving situations, people are initially unable to articulate complete requirements for problems (Fischer, Reeves, 1992). Through a means of critiquing, which reminds designers of other points of view (Miyake, 1986), people identify portions of the problem that have not yet been understood and refine the solution. In conversation, shared meanings are accrued incrementally, along with evidence of what has been understood so far (Brennan, Hulteen, 1993). People not only respond to what has been articulated immediately before, but also gradually develop a context to be coherent throughout the conversation (Stein, Thiel, 1993).

The action-reflection model represented in Figure 1 illustrates the design activity of a single designer. Applying the action-reflection model to collaborative design requires that the result of reflection be made explicit to maintain a shared understanding among the stakeholders. We call this articulation of the result of reflection a *critique*, and suggest

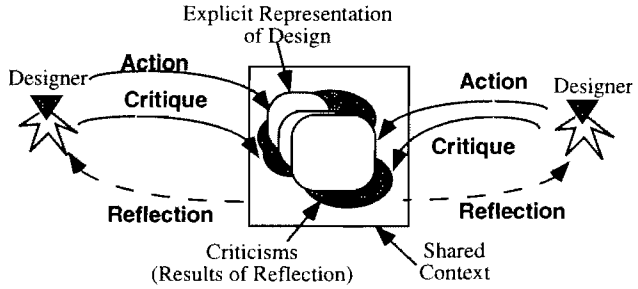

Figure 2: The Action-Reflection-Critique Model in Collaborative Design

Criticisms include pointing out possible breakdowns in the partial design and articulated knowledge as relevant to the shared context.

an action-reflection-critique model for collaborative design.

Figure 2 illustrates the action-reflection-critique model of the collaborative design task between designers. In the model, the shared context consists of representations of a partial design and criticisms including pointing out possible breakdowns in the partial design and articulated design knowledge as relevant to the shared context. Through the cyclic processes, designers both evolve a representation of their design and gradually construct and accumulate criticisms as articulated knowledge. This shared understanding helps the designers coevolve individual understanding of a problem and a solution, and increase the knowledge about the design domain.

2.3. Intertwining Knowledge Delivery and Elicitation

Our design environments are human-computer collaborative problem-solving systems based on the extension of the action-reflection-critique model. Challenges in applying the model to human-computer collaboration in design tasks have been support of communication and establishment of a shared context between a designer and a computer system.

Design representations developed within media of a design environment, which include specified requirements and a constructed solution, provide a context to be shared by designers and the design environment. Monitoring what designers have been doing with the media allows the design environment to have a partial understanding about the design task. A design environment can critique the partial design, while designers can embed their criticisms within the partial design. The evolving explicit representations of the partial design and criticisms serve as a shared understanding between designers and a design environment.

Figure 3 illustrates the model and describes how the action-reflection-critique model is applied to human-computer collaboration in design tasks. We characterize critique of a design environment as *knowledge delivery*, implying that the system delivers the information relevant to the designers' task at hand in a timely manner without explicit requests to the system. The system's critiquing (i.e., delivered knowledge) invokes designers' reflective thinking, and may make designers aware of tacit design knowledge and

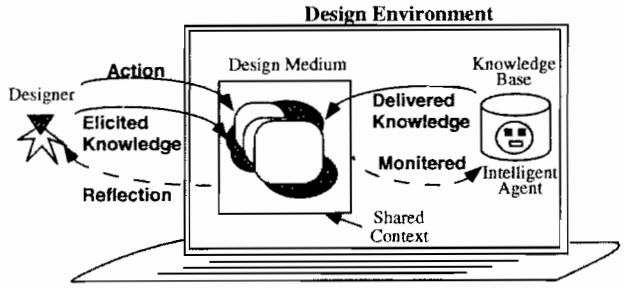

Figure 3: Applying the Action-Reflection-Critique Model to Human-Computer Collaboration

A design environment plays roles both as a design medium and as an intelligent agent that collaborates with designers. In our approach, because a design environment does not produce or modify a design representation automatically, there is no action arrow from the system. The partial design constructed in the design medium provides the shared context between the designer and the design environment. Based on the identified shared context, the system delivers the information related to the context. Designers may become aware of tacit design knowledge as relevant to the context, and store the elicited knowledge into the system as criticisms. The results of elicitation are taken into account by the system to modify its subsequent behavior.

critique back the system. We characterize the system's invoking designers to articulate criticisms to the current situation as *knowledge elicitation*. The elicited and stored knowledge is taken into account by the system to change the subsequent behavior of the system.

3. Technical Challenge

As illustrated in Figure 3, a design environment consists of two major parts: a design medium and an intelligent agent. The intelligent agent interacts with designers by delivering knowledge from a knowledge-base and changes and tunes its behavior according to the knowledge elicited by the designers during a design process. In this section, we focus our discussion on technical challenges of implementing knowledge delivery and elicitation mechanisms.

3.1. Requirements for Knowledge Delivery

Challenges for effectively implementing knowledge delivery mechanisms include how to deliver the *right knowledge* at the *right time*.

Whether a piece of knowledge is "right" can be determined in terms of the relevance of information to the current task. If the system provides a medium in which designers can represent their design, this partial design can be used to form a model of the task at hand. Using this model, the delivery mechanisms can retrieve relevant information from the knowledge-base. Determination of the relevant portions of knowledge presents a challenge because different design situations may need to view a piece of knowledge differently; thereby the definition of relevance changes dynamically. It is impossible to anticipate all possible design situations a priori (Suchman, 1987); thus a static indexing scheme for design information of the knowledge-bases is inapplicable.

Delivery mechanisms of a design environment continuously monitor user actions and intervene with them when a potential information need is detected. They are especially effective because delivered information is related to a specific condition in the current design. Intervention immediately after a suboptimal or unsatisfactory action has occurred has the advantage that the problem context is still active in the designer's mind and the designers still know how they arrived at the problematic situation (Fischer et al., 1993). At the same time, such immediate intervention should not distract designers from concentrating on the task at hand.

3.2. Requirements for Knowledge Elicitation

It is crucial that elicited knowledge is taken into account by the system to refine its subsequent behavior. In collaborative design, designers adapt their own behavior according to the evolving shared understanding (Pollack, 1985). Our experience in building a design rationale system shows that people are not motivated to articulate knowledge and store it if they do not see immediate benefit by doing so (Fischer et al., 1991).

A challenge for implementing such a knowledge elicitation mechanism is that the newly created representation must be able to be manipulated by the system. This requires designers to represent the knowledge in the system's languages — often formal and very different from designers' domain languages. Forcing designers to state their design knowledge using a knowledge representation language undermines their expressive ability. One approach to address this problem is to ask knowledge engineers to formulate their knowledge whenever they want to store elicited knowledge in the system, but this is not a feasible solution because there are risks of erroneous interpretation by knowledge engineers (Bonnardel, 1993), and it cannot be expected that knowledge engineers will be available throughout the lifetime of system usage.

Our approach is to support the evolutionary growth of a knowledge-based system through a seeding, evolutionary-growth, and reseeding cycle (see Figure 4) (Fischer et al., 1994). The system is first "seeded" by knowledge engineers in collaboration with designers, with several mechanisms that would allow designers to represent their knowledge during the use of the system. As a second phase during use, the designers store their elicited knowledge using the mechanisms, "gradually evolving" the knowledge-base. Such mechanisms must allow designers to be able to represent their thoughts, ideas, and arguments directly and distinctly using their own language. End-user modifiability (Fischer, Girgensohn, 1990) allows designers to create computer-interpretable knowledge representations without a detailed knowledge of the underlying computational mechanisms. The system is occasionally "reseeded" by knowledge-engineers in collaboration with the designers to reorganize knowledge constructed by the designers that might have introduced inconsistencies or insufficient formalisms.

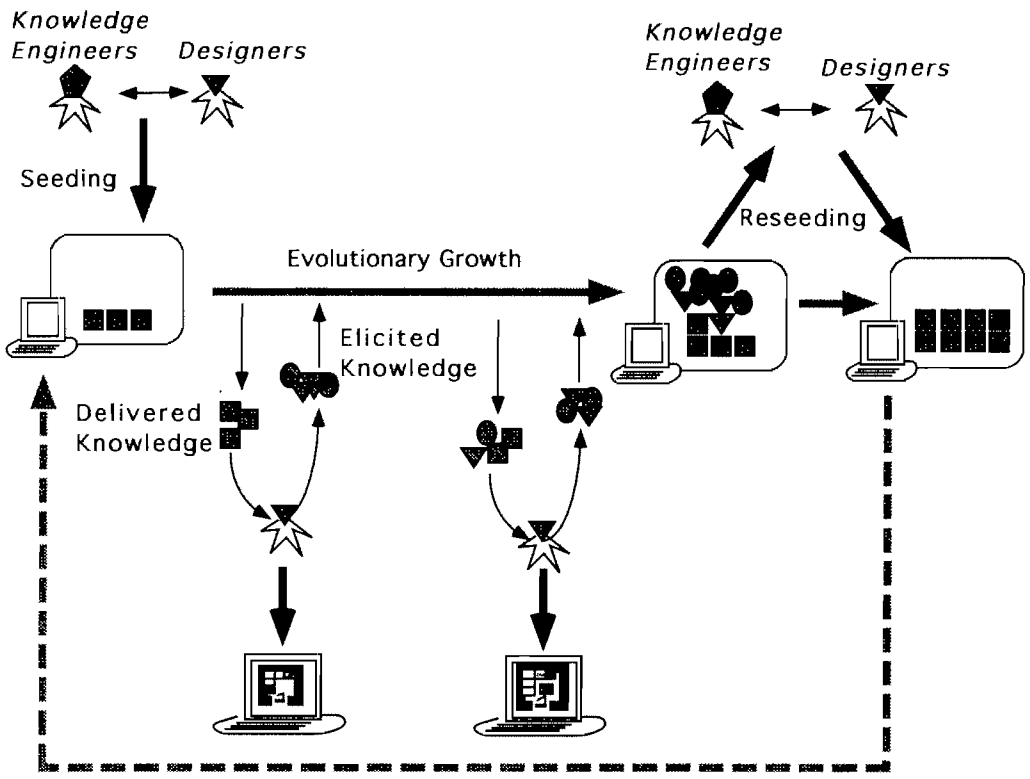

Figure 4: A Seeding, Evolutionary Growth, and Reseeding Cycle of Development Processes

Through the use of a design environment, designers gradually elicit knowledge by designing and responding to the knowledge delivery. Some part of the added knowledge may be inconsistent or not totally interpretable by the system. During the reseeding process, knowledge engineers, with the help of designers, can examine the knowledge-base and reconstruct it. Support for seeding and reseeding is discussed in Fischer et al. [1994].

4. The KID Design Environment

We have developed the KID design environment (Nakakoji, 1993) to instantiate our approach. KID is implemented in the CLOS programming language on Symbolics Genera 8.1. The design environment consists of four major components:

1. KIDSPECIFICATION, which allows designers to specify their design requirements and intentions (Figure 5);
2. KIDCONSTRUCTION, which provides designers with a palette of domain abstractions and supports them in constructing design artifacts using direct manipulation styles (Figure 6);
3. an argumentation-base, which stores design rationale represented in the IBIS structure (Conklin, Begeman, 1988) (i.e., a network of nodes, consisting of issues, answers, and arguments) (see Figure 5); and
4. a catalog-base, which stores completed floor plans (construction) together with associated specifications (see Figure 6).

The components are integrated with various delivery and elicitation mechanisms (see Figure 7). Designers can coevolve a problem and a solution by creating partial designs in KIDSPECIFICATION and KIDCONSTRUCTION. We define the representations given through the two components as "a partial design." The partial designs provide KID with a shared understanding about the designers' task at hand. KID delivers knowledge relevant to the task at hand by indicating potential breakdowns (Fischer, 1994) in the partial design and retrieving useful cases from the catalog-base for their design (Nakakoji, 1994). The delivery not only helps designers refine the partial design but also elicits tacit design knowledge from the designers by use of reminders. (Schank, 1988). Designers may store this uncovered knowledge by presenting it as argumentation or as new design objects, which results in the evolution of the system's knowledge-bases. KID dynamically takes this evolution into account and refines its behavior while supporting designers.

Through iterating this process, the designers gain an understanding of their design task as well as knowledge about the domain, and they articulate and accumulate design knowledge into the system. The system tunes its behavior and delivery according to the gradually evolving shared understanding and the elicited knowledge.

4.1. Scenario

The following scenario illustrates how a designer interacts with KID. A kitchen designer, Jeff, designs a kitchen floor plan. Jeff specifies requirements for his design task using KIDSPECIFICATION (Figure 5). Jeff starts constructing a floor plan using KIDCONSTRUCTION (Figure 6). When he puts a dishwasher on the right side of a double-bowl sink,

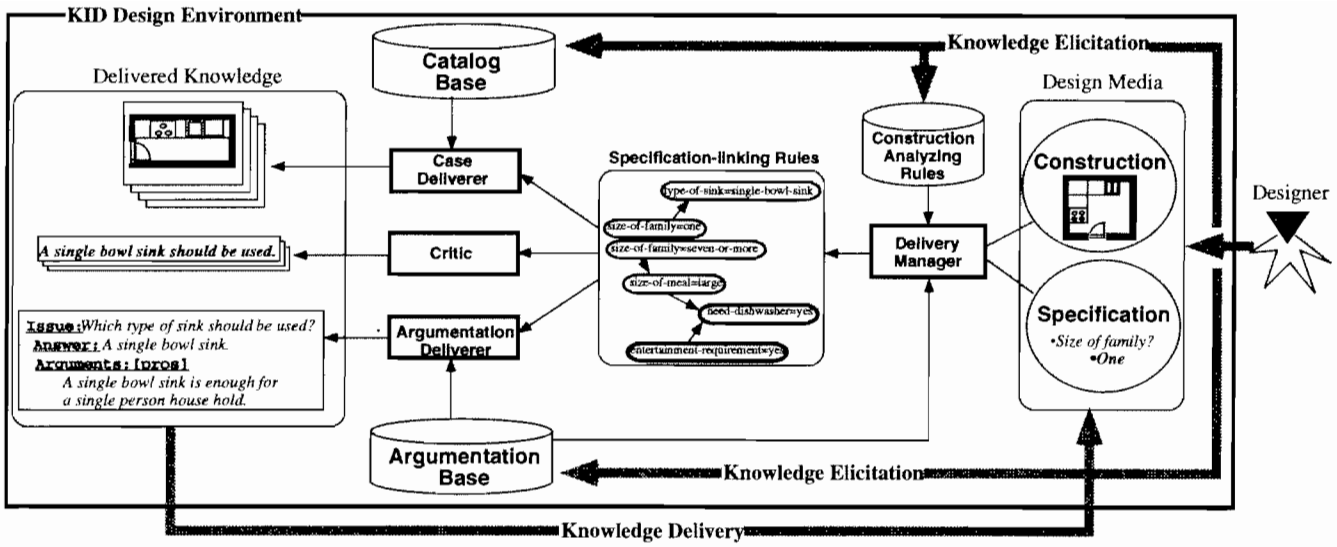

Figure 7: Knowledge Delivery and Construction in KID

A designer performs a design task through specification and construction components. The partial specification and construction provides KID with shared understanding of the task at hand. Using this understanding, a delivery manager of KID derives specification-linking rules from the argumentation-base. These rules are used by delivery mechanisms (Case Deliverer, Critic, and Argumentation Deliverer), which deliver corresponding knowledge for the designer. The designer can elicit knowledge by adding design objects to the catalog-base, by adding arguments to the argumentation-base, and by adding construction-analyzing rules using MODIFIER.

critic messages appear on the screen, one of which notifies him that he should put the dishwasher on the left side of the sink (see the *Message* window in Figure 6). Wondering why, he clicks on the critic message.

The corresponding argument is presented, describing that a kitchen should have a dishwasher on the left side of a sink because he specified that this kitchen is for a left-handed cook (see Figure 5). In fact, catalog examples, each of which has a dishwasher on the left side of the sink, have been suggested by the system in the *Catalog* window (Figure 6) to be most appropriate for the design requirements specified in KIDSPECIFICATION. Jeff reflects on the argument and thinks about specializing the kitchen for a left-handed person. Then he remembers that the resale value of the kitchen is actually a very important concern. He adds this requirement using KIDSPECIFICATION, and creates an argument that having a dishwasher on the left side of a sink may affect the resale value (see the *Argumentation* window in Figure 5). The system takes this argument into account and defines a new interdependence between the location of a dishwasher with regard to a sink and the concern for a resale value. In the *Catalog* window, the catalog examples are automatically reordered according to this newly defined interdependence, showing a catalog example that has a dishwasher on the right side of a sink on top of the list (not shown in the figures).

4.2. Architecture and Mechanisms of KID

Figure 7 illustrates the architecture and mechanisms of the KID design environment. KID delivers three types of design knowledge from its knowledge-bases:

- Catalog examples are ordered according to the appropriateness to the current specification.
- Critic messages notify potential conflicts in the floor plan in terms of the current specification and generic design rules.
- Argumentation describes alternatives to design decisions, interdependences among design decisions, and how the critic messages are related to the current specification.

KID allows designers to store elicited domain knowledge in the following manner:

1. to add arguments to the argumentation-base;
2. to add, modify, or delete questions and answers in KIDSPECIFICATION;
3. to store completed specifications and constructions as a catalog example;
4. to add, modify, or delete palette items in KIDCONSTRUCTION; and
5. to define or redefine construction-analyzing rules.

Delivery mechanisms and some elicitation mechanisms (above 1 and 2) use *specification-linking rules*. These rules represent interdependences among design decisions and are used to infer the relevance between different types of design representations. The specification-linking rules are further described below. Other elicitation mechanisms (above 3, 4 and 5) are supported differently and described in detail in Nakakoji [1993] and in Girgensohn [1992]. The *Store into Catalog* command allows designers to store their specification and construction into the catalog-base, which is reconstructed on the fly. This allows designers to store several versions of their design as catalog examples. The MODIFIER system (Girgensohn, 1992) allows designers to create or modify design objects of KIDCONSTRUCTION, including palette items and construction-analyzing rules. MODIFIER provides a property sheet interface to designers

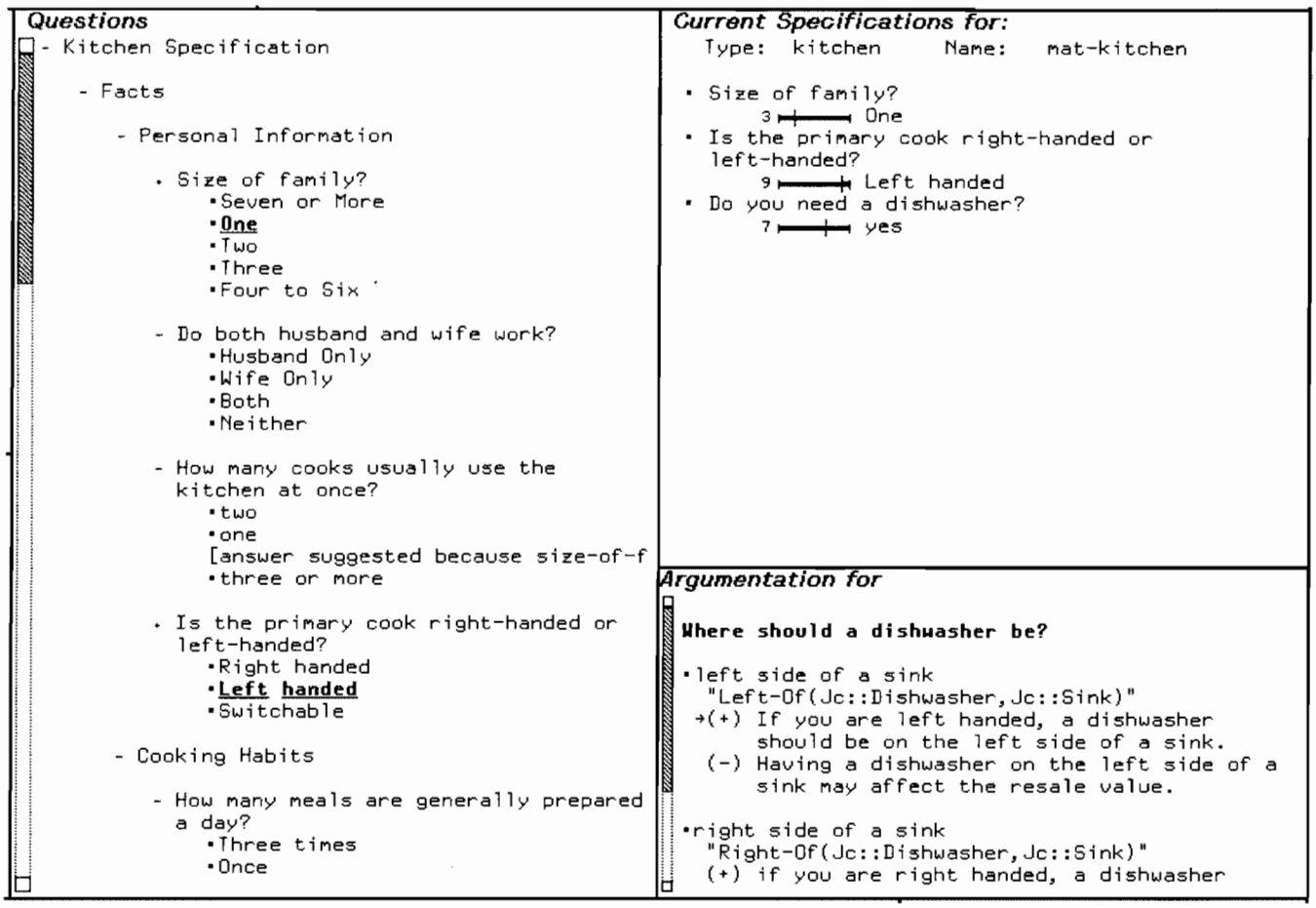

Figure 5: KIDSPECIFICATION

Designers can select answers presented in the *Questions* window. The summary of currently selected answers appears in the *Current Specifications for* window. Each answer is accompanied by a slider that allows designers to assign a weight representing the relative importance of the answer (e.g., most importance to the left-handed cook requirement (i.e., 9) and little importance to the single-person household requirement (i.e., 3)). The *Argumentation for* window provides further explanation about how a presented critic message (i.e., a location of a dishwasher with regard to a sink) (see Figure 6) is related to the current specification (i.e., one of the selected answers - a left-handed cook), as well as alternatives for the location of a dishwasher including the one just created.

for a design object that is internally represented as a CLOS object. MODIFIER helps designers to edit the attributes of the object by making suggestions and providing help messages.

Representation of Specification-Linking Rules. Each specification-linking rule is derived from an argument stored in the argumentation-base. The argumentation-base consists of a network of nodes of issues, answers and arguments, each of which is implemented as a CLOS object. A class `node`, which has subclasses `issue`, `answer`, and `argument`, provides methods necessary for dealing with display and maintaining links between nodes. Figure 8 shows definitions of the issue, answer, and argument about the location of a dishwasher in terms of a sink, which corresponds to the presentation in the *Argumentation-for* window in Figure 5. Property sheet interfaces are provided for designers to define each of these objects (see Figure 9).

The underlying formalism of the argumentation-base is a network structure of nodes, consisting of an issue, optional answers, and associated pro or con arguments. In order to form a structural network among the questions, each question must have at least one super issue. The root issue is called *"Kitchen Specification."* Currently, the root has three subtrees, *"Facts," "Preferences"* and *"Feature Constraints." "Facts"* contains the *"Personal* Information," *"Cooking Habits,"* and *"Other Activities"* subtrees. These structures can be modified on the fly by designers by editing an issue object definition using a property sheet interface.

Each question and answer has a unique name as an identifier. For example, the question *"Where should a dishwasher be?"* has a name *"where-dishwasher,"* and the answer *"left side of a sink"* has a name *"left-side-of-a-sink."* A pair of the identifiers of an issue and answer is called a domain-distinction, which constitutes vocabulary over the domain (Winograd, Flores, 1986).

An argument object has the *Related Domain Distinction* slot (see Figures 8 and 9) that relates this argument to another issue-answer pair, and dynamically infers the dependency among the two issue-answer pairs (i.e., *specification-linking rules*), as described below. When designers want to relate an argument to another question,

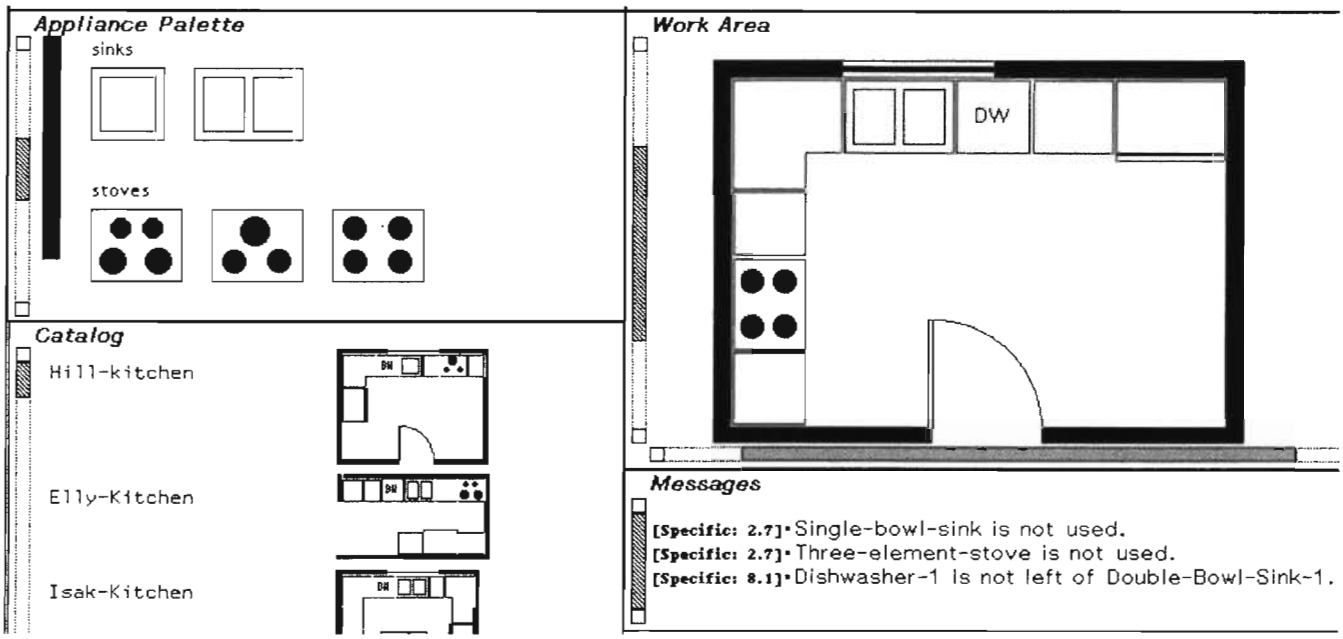

Figure 6: KIDCONSTRUCTION

Designers construct a kitchen floor plan in the *Work Area* using a direct manipulation style to select and place design units from the appliance palette. Designers may copy an example from the *Catalog* window, where catalog examples are presented in the order of accordance with the current specification (see Figure 5). The *Messages* window presents critiquing messages that are detected by KID. Numbers indicate computed relative importance of each critiquing message in terms of the current specification.

```
(defobject where-dishwahser issue
  :name "where-dishwasher"
  :superissues  '("feature-constraints")
  :subissues nil
  :text-description "Where should a dishwasher be?"
  :type :single-value
  :answers '("left-side-of-a-sink"
             "right-side-of-a-sink"
             "near-wall-cabinet")
  :condition t)
(defobject left-side-of-a-sink answer
  :name "left-side-of-a-sink"
  :issue "where-dishwasher"
  :text-description "Left side of a sink"
  :critic-rule "Left-Of(Dishwasher, Sink)"
  :arguments '(left-side-of-a-sink-arg-1))
(defobject left-side-of-a-sink-arg-1 argument
  :text-description "If you are left handed, a dishwasher should be on the left side of a sink"
  :answer "left-side-of-a-sink"
  :polarity :pro
  :author "Nakakoji, Kumiyo"
  :Time 2938902977
  :related-domain-distinction (eq "left-right" "left"))
```

Figure 8: Issue, Answer, and Argument Objects in the Argumentation-Base

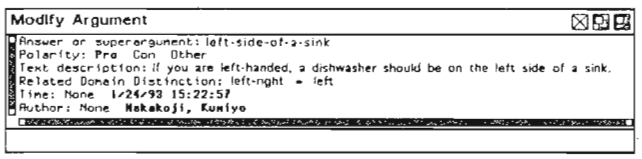

Figure 9: A Property Sheet Interface for Modifying the Argument Object

A property sheet for modifying or creating an argument.

hitting the *HELP* key triggers KIDSPECIFICATION to show all the names of existing issues (Figure 10). Designers can pick one of them, followed by either "=" or "≠", followed by the name of one of the related answers.

Derivation of a Specification-Linking Rule. Specification-linking rules are a subset of "serve relationships" (Fischer et al., 1991) that can be defined over the argumentation base. The serve relationships represent

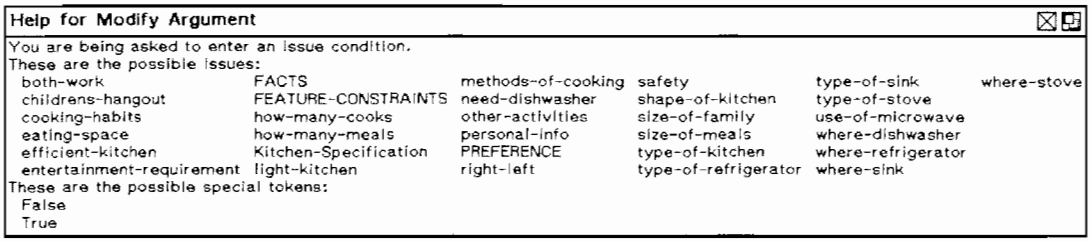

Figure 10: Existing Issue Names

A list of names of currently existing issues. When designers create a new issue using a property sheet, the defined name is automatically added to this list.

which issue resolution affects the resolution of other issues. For example, if you answer that the size of the household is one, then the meal size suggested is small. Specification-linking rules represent this dependency, namely, a certain issue resolution determines how to answer other issues. Some answers are related to features in the representation of constructions. In short, those rules form a decision-making network, and some of the decisions are related to surface features in the partial construction.

KID automatically derives such a serve relationship between two issue-answer pairs from an argument of the argumentation-base. Often, an argument to an issue-answer pair `X` is related to another issue-answer pair `Y`, and this association is made by designers filling in the *related-domain-distinction* field using the property sheet (Figure 9). If the argument is a pro argument, the relationship is an implication: `Y implies X`. If the argument is a con argument, the relationship is an implication of the negation of the issue-answer pair: `Y implies not X`. For example, the argument *"If you are left-handed, a dishwasher should be on the left side of a sink"* is related to the issue-answer pair *"left-right=left"* and because the argument is a pro argument for the answer *"Where should a dishwasher be?: Left side of a sink,"* the specification-linking rule *"left-right=left"* → *"where-dishwasher=left-side-of-a-sink"* is derived.

The system dynamically identifies this interdependence between two issue-answer pairs and derives a specification-linking rule. Each time designers select an answer to an issue (`Y`), the system scans all the arguments to identify issue-answer pairs `X` that have arguments that are associated with the selected issue-answer pair, `Y`. Then, according to its polarity, pro or con, the system defines a PROLOG term "`require(X Y)`" or "`require(X (not Y))`." For example, from the example above, the following term is defined:

```
require((EQ left-right left)
        (EQ where-dishwasher
            left-side-of-a-sink))
```

The system uses the PROLOG engine to perform inference, and the maximum number of steps of inference can be determined by designers (the default is 3). This way, a newly added argument is immediately reflected in the derivation of the specification-linking rules.

Knowledge Delivery in KID. The *Delivery Manager* of KID monitors KIDCONSTRUCTION and KIDSPECIFICATION and reformulates the interdependence network represented by specification-linking rules each time a modification is made in the two components. Because KIDSPECIFICATION has been built as a hypertext interface to the argumentation-base, selecting answers in KIDSPECIFICATION is reflected in reorganizing the specification-linking rules. In order to reflect the current construction to the formation of the specification-linking rules, the delivery manager consults with the construction-analyzing rules that can parse the *Work Area* of KIDCONSTRUCTION and identify notable characteristics, such as whether a dishwasher is on the left side of a sink.

The interdependence network formulated from specification-linking rules is used by three mechanisms to deliver three types of design knowledge. The *Case Deliverer* mechanism applies the rules to each catalog example, computes an appropriateness value for each example, and orders the examples in the *Catalog* window by the computed values. The *Critic* mechanism applies the rules to the current construction, detects conflicts in the construction in terms of the rules, computes the importance of each conflict in terms of the current weighted specification, and presents critiquing messages with a number indicating the importance in the *Message* window. Finally, the *Argumentation Deliverer* mechanism maintains the derivation of specification-linking rules from the argumentation-base, and presents related arguments in the *Argumentation* window when designers request further explanation for the presented critiquing messages. The mechanisms are described in detail in Nakakoji [1993].

5. User Study

KID has been evaluated by observing ten subjects, including both domain-experts and novices, using the system. The *constructive interaction method* (Miyake, 1986) was used to analyze the interaction between the subjects and the system. The method uses two subjects in each group and observes them talking to each other. The subjects were asked to design a kitchen floor plan, given the informal requirement description shown in Figure 11. Test sessions were videotaped, and the protocols were partially transcribed.

Figure 12 summarizes types of reactions made by the subjects when KID presented critics. The classification in the figure is simplified for the sake of clarity. When design knowledge was delivered, subjects responded in one of the following ways: (1) they applied the delivered knowledge to refine the partial design, (2) they explored the related information space to the delivered knowledge, or (3) they articulated new design knowledge by arguing against the

Initially Given Specifications:
- Both the wife and husband have jobs.
- During weekdays, the wife mostly cooks.
- During weekends, the couple cooks meals together.
- The wife is left-handed.

Hidden Requirements:
- The couple lives with five children.
- They come home for lunch. Therefore, they prepare meals three times a day in the kitchen.
- They often entertain.
- They do not need an eating space.
- They need a dishwasher.
- They need a double-bowl sink.
- Safety and efficiency are important factors for them.

Figure 11: The Requirement Description of the Task

In order to observe how subjects understand new aspects of the design problem, some of the conditions were initially hidden. These conditions were revealed to the subjects as requested. Anything asked that was not stated in the conditions was answered as unknown.

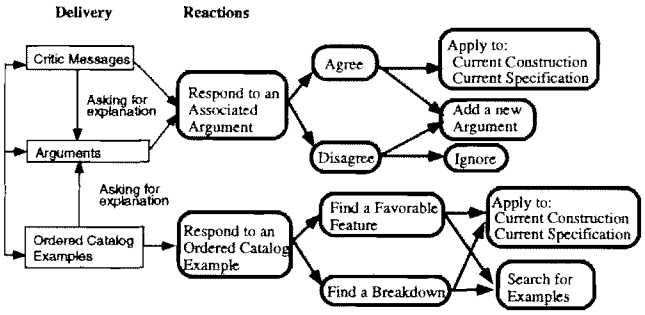

Figure 12: Subjects' Reactions to the Critics

The subjects' reactions to the system's knowledge delivery are classified. When they were presented with the associated argument through a critic message, the subjects either agreed or disagreed with the argument, and different reactions were made accordingly. When ordered catalog examples were presented, the subjects either liked or disliked one of the examples, depending on whether they discovered a good or unknown feature or an unfavorable feature in the example. When they accessed arguments explaining how and why the catalog examples are related to their current task, the reaction was the same as that of the critic messages discussed above.

delivery.

Active knowledge delivery was often found to trigger the subjects to reflect on their partial design. Subjects often discovered breakdowns or important considerations about which they had not previously been aware, and they often reacted to delivered knowledge and argued against it in terms of their task at hand.

An unexpected benefit of knowledge delivery was that it encouraged the subjects to explore the system's information space. There is evidence that people search longer for answers to questions when they believe they know the answer (Reder, Ritter, 1992). High *feelings of knowing* correlate with longer search times. When KID delivered information that was relevant to the task at hand but not quite right, the subjects then gained this "feeling of knowing," which made their information search longer.

When presented with a critic message, the initial reaction of the subjects was often to ask for further explanations of why the message was significant and how it was related to their specification. For example, when KIDCONSTRUCTION displayed the critic message *"the dishwasher should not be used,"* subjects did not discuss this initial critic message, but instead they examined the underlying argumentation of the message, which was displayed in the *Argumentation* window, by clicking on the initial critic message. When the subjects disagreed with the argument, they often articulated counterarguments. When the subjects attended to a catalog example that was on top of the ordered list in the *Catalog* window in KIDCONSTRUCTION, they responded either by discovering favorable or problematic features in the example, or by wanting to know why and how the example was relevant to their current specification.

In what follows, we focus on the three types of user reactions and effects of knowledge delivery, and illustrate each with transcripts collected through the user studies.

Reframing a Problem Specification. Often the subjects found a design concept of which they had not been aware by being reminded by a delivered catalog example. For example, in the following transcript, the subject was given the requirement description (Figure 11), and he had not been aware of the concept of *"family size"* until he read the specification of *Dreyer-Kitchen*, a delivered catalog example. Then, he realized that the concept is important for the design task and developed the specification after asking the question.

> **[1]:** [*Dreyer-Kitchen* was presented on top of the catalog example list. The subject was reading the specification of the example.] .. *both work, seven family ... umm... We do not know how many family she* [the hypothesized client] *has ... we just don't have enough information about the size of family, and the size might grow, so.... Can I ask you the size of family now?*

Reframing a Solution Construction. The system's delivered knowledge sometimes helped the subjects to refine their partial solution. The below transcripts illustrate one of many examples that were observed during the study.

> **[2]:** [The system presented a critic *"a double-door refrigerator is not used."*]
>
> *Why so ...?*
>
> [The subject asked for the explanation, and found the argumentation *"if you often use a microwave, you probably need to store a lot of frozen food, therefore you need a big refrigerator."*]
>
> *Oh, because I answered I need a microwave* [in KIDSPECIFICATION]. *OK, then probably you need to change that* [in KIDCONSTRUCTION]. *That's a good point. Actually..*

In this transcript, the subject agreed with the critic about the type of refrigerator in terms of the specification of the use of a microwave oven. Then, he replaced the single-door refrigerator of his floor plan with a double-door refrigerator.

Articulating a New Argument. The subjects not only learned delivered domain knowledge but also articulated counter opinions against the delivered knowledge. The following three portions of scripts illustrate how they articulated previously tacit knowledge. Some of the elicited knowledge was incorporated into the system using the property sheet interfaces.

> **[3]:** [when the system suggested having an L-shaped kitchen for two cooks and the subject read an argument for the L-shaped kitchen] *I see ... Let's check out the argument for an island kitchen and what it said about two cooks kitchen ... No...? Umm.. well, could I create another argument?* [the partner invoked the property sheet to create a new argument] *OK. An island kitchen is good for two cooks because it provides additional counter space ... well.. Two or more (cooks) actually....*

In the transcript, the subject did not disagree with the L-shaped critic, but was motivated by the argument to explore the related argumentation space by looking for alternatives. Consequently, she remembered a more interesting argument for the island kitchen, and added the argument into the system.

> **[4]:** [a critic fired *"a refrigerator should be close to a door"* so that *"incoming cold food can quickly be stored in the refrigerator"*] *No, because a garage is not next to a kitchen, right? You have to carry groceries anyway.*
>
> **[5]:** [the same critic fired] *all right ... but this way* [pointing to their current construction], *you can come in and put them on the counter, and put it in the refrigerator. So let's add an argument to this ...*

This type of reaction was most often found. When the subjects disagreed with a presented argument, the subjects often articulated arguments against the argument. The refrigerator-door critic rule was disputed by many subjects. The subjects came up with many arguments against the incoming food argument. Some of such counterarguments were generally discussed (i.e., transcript **[4]**), and others were discussed in terms of their current construction (i.e., transcript **[5]**).

6. Discussion

The results of the user studies have demonstrated that collaboration between designers and the design environment was supported through a series of delivery and elicitation of design knowledge. At the same time, the studies have uncovered many future research issues including:

- *support for providing perspectives:* The subjects sometimes wanted to apply totally different design knowledge without changing the underlying knowledge-base; for example, one wanted to apply specific design knowledge for a Japanese style kitchen. The knowledge-base of the system should be partitioned according to multiple perspectives that designers can choose for individual design tasks (Nakakoji, Sumner, Harstad, 1994).
- *support for designers to find appropriate domain* distinctions: currently, a list of existing names for issues is provided in an alphabetical order, but the subjects had trouble finding it.
- *support for annotating design:* some of the subjects wanted to directly annotate their partial design rather than recording arguments in a separate interface.
- *support for using the complex design environment:* although the effectiveness of KID comes from the integration, the system currently is so complex that the subjects often got lost. Mechanisms such as intelligent agents are necessary to guide designers in using KID.
- *support for consistency checking:* currently, KID allows designers to make any modification to the system's knowledge-base, assuming that possible inconsistencies could be checked during the reseeding phase. In order to prevent unnecessary iterations, the system should help designers when making modifications regarding obvious inconsistencies of knowledge-bases.

Despite these findings, the results of the studies have assured us that our architecture will provide an effective framework for supporting human-computer collaboration in design tasks. In what follows, we briefly discuss how our design environment approaches address other research challenges.

Information Overload Problems. The large and growing discrepancy between the amount of potentially relevant information and the amount any one designer can know and remember limits the ability of designers to take advantage of high-functionality knowledge-based design environments (Fischer, Henninger, Nakakoji, 1992). Designers may not know what information they need, how to ask for that information, or that potentially useful information exists in the system. The knowledge delivery mechanisms of KID address this problem. The system uses the partial design represented in the system to identify the designer's information needs and to deliver information relevant to the task at hand. The partially identified current task through KIDSPECIFICATION and KIDCONSTRUCTION can be used as queries submitted to information retrieval mechanisms, providing the background context for an information search (Fischer, Nakakoji, 1991).

Knowledge Acquisition Problems. People are good at responding to a given situation, but it is difficult for domain experts to start articulating domain knowledge given no context. Delivering knowledge by a design environment stimulates designers to elicit tacit knowledge (Brown, Duguid, 1992). In the user study, we have observed that the subjects successfully articulated their criticisms to what the system had done. They often disagreed with the rationale behind KID's delivery, and started to articulate argumentation. They often wanted to store the arguments into the system and to define a new interdependence between design concepts "to see what would happen."

Long-term Indirect Collaboration Among Designers. An argumentation-base and catalog-base of a design environment facilitate long-term indirect communication among designers (Fischer et al., 1992). Designers communicate through accrued design representations in a design environment. A portion of argumentation articulated and stored by a designer can be used to produce a specification-linking rule, which helps other designers at

other times to solve other design problems. Designers store a design into a catalog-base, which helps other designers to produce ideas for a solution. Thus, a design environment that intertwines knowledge delivery and elicitation supports designers to naturally carry out design as an argumentative process (Rittel, Webber, 1984) among designers over a long period of time.

Thought-provoking Mechanisms. It is noteworthy that delivering *not-quite-right* knowledge may also help designers. We observed during the user study that even when designers found the delivered information to be of little relevance, they often were willing to attend to the delivered knowledge and modify the knowledge-base by adding new design principles or concepts in response to it, leading to knowledge elicitation. A computational agent does not always have to provide the *right* answer to the problem at hand; it can be thought provoking, as a novice's critiquing enhanced an expert's performance in Miyake's study (Miyake, 1986). The system's delivery of both "right" and "not-quite-right" knowledge reminds designers of potentially important design information, and invokes them to uncover their tacit knowledge.

7. Conclusion

The specification and construction components embedded in KID provide the shared understanding between designers and the system. In our work, shared understanding is used to identify the designers' information needs, to locate relevant information to support the designers' task at hand, and to deliver the information to the designers. By having shared understanding, knowledge delivery mechanisms are more tuned toward delivering the right knowledge at the right time. The delivered knowledge helps designers uncover breakdowns in a partially represented design, provokes designers in their reflective thinking, and helps them to become aware of tacit design knowledge and to articulate it. The elicited knowledge from designers contributes to the evolution of a knowledge-base, and the system tunes their subsequent behavior based on the evolving shared understanding.

We have presented a model for supporting human-computer collaborative design and discussed an architecture for the model. The study of KID, which has been built based on the architecture, has demonstrated that intertwining knowledge delivery and elicitation is critical for supporting human-computer collaboration.

Acknowledgments

We thank the HCC group at the University of Colorado, who contributed to the conceptual framework and the systems discussed in this paper. We also thank Barbara Gibbons of Kitchen Connection at Boulder, Colorado, for her valuable time and her comments on our work. We thank Loren Terveen of AT&T Bell Labs, for his comments and suggestions on an earlier version of the paper. The research was supported by the National Science Foundation under grants No. IRI- 9015441 and MDR-9253425; Software Research Associates, Inc. (Tokyo); the Colorado Advanced Software Institute; US WEST Advanced Technologies; and NYNEX Science and Technology Center.

References

N. Bonnardel 1993. Expertise Transfer, Knowledge Elicitation, and Delayed Recall in a Design Context, *Behaviour and Information Technology*, 12(5):304-314.

S.E. Brennan, E.A. Hulteen 1993. Interaction and Feedback in a Spoken Language System, in Working Notes of the AAAI Fall Symposium Workshop on Human-Computer Collaboration: Reconciling Theory, Synthesizing Practice, AAAI, 1-5, Raleigh, NC.

J.S. Brown, P. Duguid 1992. *Enacting Design for the Workplace*, in P.S. Adler, T.A. Winograd (eds.), *Usability: Turning Technologies into Tools*, Oxford University Press, New York, NY, 164-197.

J. Conklin, M. Begeman 1988. gIBIS: A Hypertext Tool for Exploratory Policy Discussion, *Transactions of Office Information Systems*, 6(4), October:303-331.

G. Fischer 1994. *Turning Breakdowns into Opportunities for Creativity*, in E. Edmonds (ed.), *Creativity and Cognition*, Penrose Press, (in press).

G. Fischer, A.C. Lemke, R. McCall, A. Morch 1991. Making Argumentation Serve Design, *Human Computer Interaction*, 6(3-4):393-419.

G. Fischer, J. Grudin, A.C. Lemke, R. McCall, J. Ostwald, B.N. Reeves, F. Shipman 1992. Supporting Indirect, Collaborative Design with Integrated Knowledge-Based Design Environments, *Human Computer Interaction, Special Issue on Computer Supported Cooperative Work*, 7(3):281-314.

G. Fischer, K. Nakakoji, J. Ostwald, G. Stahl, T. Sumner 1993. Embedding Critics in Design Environments, *The Knowledge Engineering Review Journal*, 8(4), December:285-307.

G. Fischer, R. McCall, J. Ostwald, B. Reeves, F. Shipman 1994. Seeding, Evolutionary Growth and Reseeding: Supporting Incremental Development of Design Environments, in Human Factors in Computing Systems, CHI'94 Conference Proceedings (Boston, MA), 292-298, ACM.

G. Fischer, A. Girgensohn 1990. End-User Modifiability in Design Environments, in Human Factors in Computing Systems, CHI'90 Conference Proceedings (Seattle, WA), 183-191, ACM, New York.

G. Fischer, S. Henninger, K. Nakakoji 1992. *DART: Integrating Information Delivery and Access Mechanisms*, Unpublished Manuscript.

G. Fischer, K. Nakakoji 1991. 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, 67-73, Cambridge, MA.

G. Fischer, K. Nakakoji 1992. Beyond the Macho Approach of Artificial Intelligence: Empower Human Designers - Do Not Replace Them, *Knowledge-Based Systems Journal*, 5(1):15-30.

G. Fischer, B.N. Reeves 1992. Beyond Intelligent Interfaces: Exploring, Analyzing and Creating Success Models of Cooperative Problem Solving, *Applied Intelligence, Special Issue Intelligent Interfaces*, 1:311-332.

J.S. Gero (editor) 1989. *Artificial Intelligence in Design*, Springer-Verlag, Berlin, Germany.

A. Girgensohn 1992. *End-User Modifiability in Knowledge-Based Design Environments*, Ph.D. Dissertation, Department of Computer Science, University of Colorado, Boulder, CO, Also available as TechReport CU-CS-595-92.

N. Miyake 1986. Constructive Interaction and the Iterative Process of Understanding, *Cognitive Science*, 10:151-177.

K. Nakakoji 1993. *Increasing Shared Understanding of a Design Task Between Designers and Design Environments: The Role of a Specification Component*, Unpublished Ph.D. Dissertation, Department of Computer Science, University of Colorado, Also available as TechReport CU-CS-651-93.

K. Nakakoji 1994. *Case-Deliverer: Retrieving Cases Relevant to the Task at Hand*, in *Lectures Notes in AI: EWCBR-93*, Springer-Verlag, Kaiserslautern, Germany, (in press).

K. Nakakoji, T. R. Sumner, B. Harstad 1994. Perspective-Based Critiquing: Helping Designers Cope with Conflicts Among Design Intentions, in J. Gero (ed.), in Artificial Intelligence in Design'94, Butterworth-Heinemann Ltd, Lausanne, Switzerland, (forthcoming).

M. Polanyi 1966. *The Tacit Dimension*, Doubleday, Garden City, NY.

M.E. Pollack 1985. Information Sought and Information Provided: An Empirical Study of User/Expert Dialogues, in Human Factors in Computing Systems, CHI'85 Conference Proceedings (San Francisco, CA), 155-159, ACM, New York.

L.M. Reder, F.E. Ritter 1992. What Determines Initial Feeling of Knowing? Familiarity With Question Terms, Not With the Answer, *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 18(3).

H.W.J. Rittel, M.M. Webber 1984. *Planning Problems are Wicked Problems*, in N. Cross (ed.), *Developments in Design Methodology*, John Wiley & Sons, New York, 135-144.

R.C. Schank 1988. Remindings and Memory: Dynamic Memory: A Theory of Reminding and Learning in Computers and People (Chap 2), in J. Kolodner (ed.), in Proceedings: Case-Based Reasoning Workshop, Morgan Kaufmann Publishers, 1-16, Clearwater Beach, FL.

D.A. Schoen 1983. *The Reflective Practitioner: How Professionals Think in Action*, Basic Books, New York.

H.A. Simon 1981. *The Sciences of the Artificial*, The MIT Press, Cambridge, MA.

A. Stein, U. Thiel 1993. A Conversational Model of Multimodal Interaction in Information Systems, in Proceedings of AAAI-93, Eleventh National Conference on Artificial Intelligence, AAAI Press/The MIT Press, 283-288, Washington, DC.

L.A. Suchman 1987. *Plans and Situated Actions*, Cambridge University Press, Cambridge, UK.

T. Winograd, F. Flores 1986. *Understanding Computers and Cognition: A New Foundation for Design*, Ablex Publishing Corporation, Norwood, NJ.