

Facilitating collaborative design through representations of context and intent

G. Fischer¹, K. Nakakoji² and J. Otswald¹

¹Department of Computer Science and Institute of Cognitive Studies

University of Colorado

Boulder Colorado 80309-0430

USA

²Software Engineering Laboratory

Software Research Associates, Inc.

1-1-1 Hirakawa-cho, Chiyoda-ku

Tokyo 102

Japan

Abstract

Domain-oriented design environments require support for three types of collaboration: (1) collaboration between domain-oriented designers (the users of design environments) and design environment builders, (2) collaboration between domain-oriented designers and clients (users of designed individual artifacts using a design environments), and (3) long-term indirect collaboration among designers. Design environments provide representations that serve as a shared context for collaboration and ground languages of design.

In this paper, we describe two components of our research work exploring domain-oriented design environments. First, the Knowing-in-Design (KID) environment uses explicit representations of the designers' task at hand (representing a partial articulation of their intent) to facilitate mutual education between clients and designers, and to deliver design knowledge relevant to the task at hand. Second, the Evolving-Artifact-Approach (EVA) which uses descriptive representations, functional representations, and seed prototypes to facilitate the mutual education process between designers and design environment builders necessary for achieving a deep knowledge of the application domain, for capturing design rationale, and for representing domain knowledge.

Domain-oriented design environments are evolving artifacts supporting long-term indirect collaboration between designers: Each design produced in design environments contributes to the accumulated design knowledge. By delivering relevant information from the knowledge base, design environments create a virtual collaboration with past designers.

Keywords: domain oriented design environments, long-term indirect collaboration, coopera-

tive problem-solving systems, shared context, mutual education, languages of doing, situational interpretation, focus of attention, EVA, evolving artifact, KID, communication of intent, human-centered intelligent agents, computer-supported cooperative work, information overload, information access and delivery, information filtering, learning, institutional memory, design rationale.

1 Introduction

We view design as intrinsically collaborative and ongoing. Complexity in design arises from the need to synthesize different perspectives on a problem, the management of large amounts of information potentially relevant to a design task, and understanding the design decisions that have determined the long-term evolution of a designed artifact. Our approach to supporting design with computers focuses on human-centered design support with domain-oriented design environments. Rather than modeling the cognitive processes of designers, we augment the abilities of designers to understand, manage and communicate complexity.

In this paper we first describe two *modes* of collaborative design and discuss how design environments support two of collaboration at two *levels* of design: the design of individual artifacts and the design of design environments. The KID design environment is described to illustrate support for collaboration in the design of individual artifacts, and the EVA approach to design environment building is describe to illustrate collaboration in the context of system building.

2 Two Modes of Collaboration

Design environments support two modes of collaborative design: (1) Short-term, Directed Collaboration; and (2) Long-term, Indirect collaboration. These modes are actually two points along a continuum of possible collaboration paradigms, but for the purposes of clarity we will discuss them as if they were distinct.

Short-term, Directed Collaboration. Design requirements originate not from designers but from clients, who do not have the knowledge necessary to implement a solution. Initially, there is a symmetry of ignorance between design designers and clients, in which 1) the understanding required to solve the design problem is distributed, and 2) there is no common language that allows the stakeholders to communicate their understanding to each other. Mutual education processes enable the construction of a shared “language of doing” that allows clients and designers to collaboratively build the knowledge required to solve the design problem.

In short-term, directed collaboration, designers and clients work together to define the problem to be solved as well as to produce a solution. Our primary emphasis is on conceptual coordination rather than on physical coordination. We are interested in creating a shared understanding among stakeholders that allows both clients and designers to contribute their respective knowledge to the design task. Complex problems can be only vaguely understood before the design process begins. Attempts at producing a solution reveal additional design requirements that could not be foreseen. For complex design problems, design requirements and solutions must coevolve.

Long-term, Indirect Collaboration among Designers. Long-term, indirect collaboration is required in the design of complex and evolving artifacts, such as local area network design

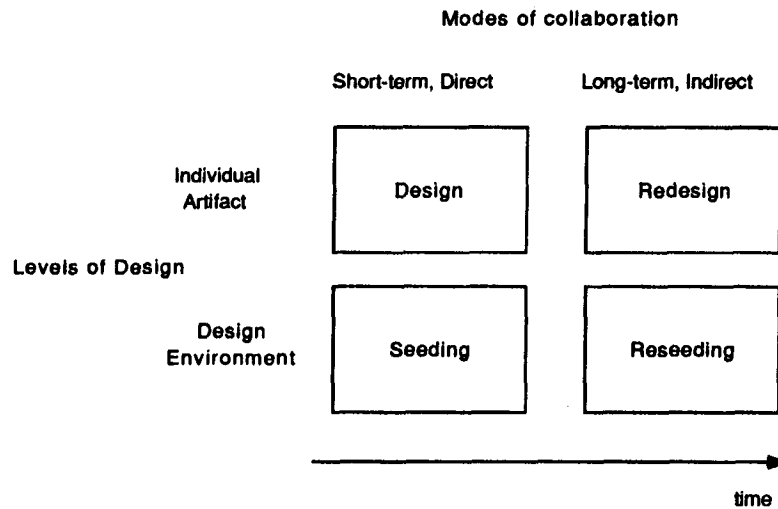


Figure 1: Four Collaborative Design Processes

The left two cells of the matrix describe face-to-face direct collaboration. In seeding, domain experts and system builders work together to design the design environment. The box labeled design represents a collaboration between domain experts and clients. The right two cells of the matrix represent indirect collaboration, where design knowledge stored in the artifact facilitates virtual collaboration with past designers.

and software design, which are maintained and modified over a span of years. In ongoing design, the emphasis must be on sharing knowledge among designers rather than on knowledge acquisition by individuals. Designers who maintain and modify complex and evolving artifacts collaborate indirectly with the original designers.

The term, “indirect collaboration” implies that communication occurs through a shared knowledge space rather than directly between designers. A shared knowledge space that supports indirect collaboration provides a background, or tradition, against which the design of individual artifacts takes place. Knowledge spaces that support indirect collaboration must be dynamic because design traditions are constantly evolving as new knowledge is discovered and old knowledge becomes obsolete.

Design documentation, reference materials, precedent solutions, and design guidelines are information sources that enable indirect, long-term collaboration with past designers. However, the existence of large amounts of design information does not guarantee effective collaboration. In information-intensive domains, the challenge is not merely to accumulate information, but rather it is to find information relevant to unanticipated problems that arise during the design task.

3 Two Levels of Design in Design Environments

There are two interrelated and ongoing design processes addressed in our design environment work: (1) the design of individual artifacts within design environments; and (2) the design of design environments themselves. Our claim is that both design processes are collaborative, are anchored by representations, and employ languages of doing.

Figure 2 illustrates the levels of collaborative design involved in building and using design environments. Three types of people are involved: (1) *design environment builders*; (2) *designers*, who use the design environment; and (3) *clients*, who use individual design artifacts.

Although it is vital to facilitate the communication among people in the two adjacent levels (see Figure Figure 2), the goal is to reduce the amount of communication required between people in non-adjacent levels. People in each level use different *languages* for achieving their design. Communication distance represented by the number of levels involved represents the number of languages that the participants have to deal with. The more languages, the larger the risk that mis-communication may take place. We have to reduce the problem of miscommunication by reducing the communication distance.

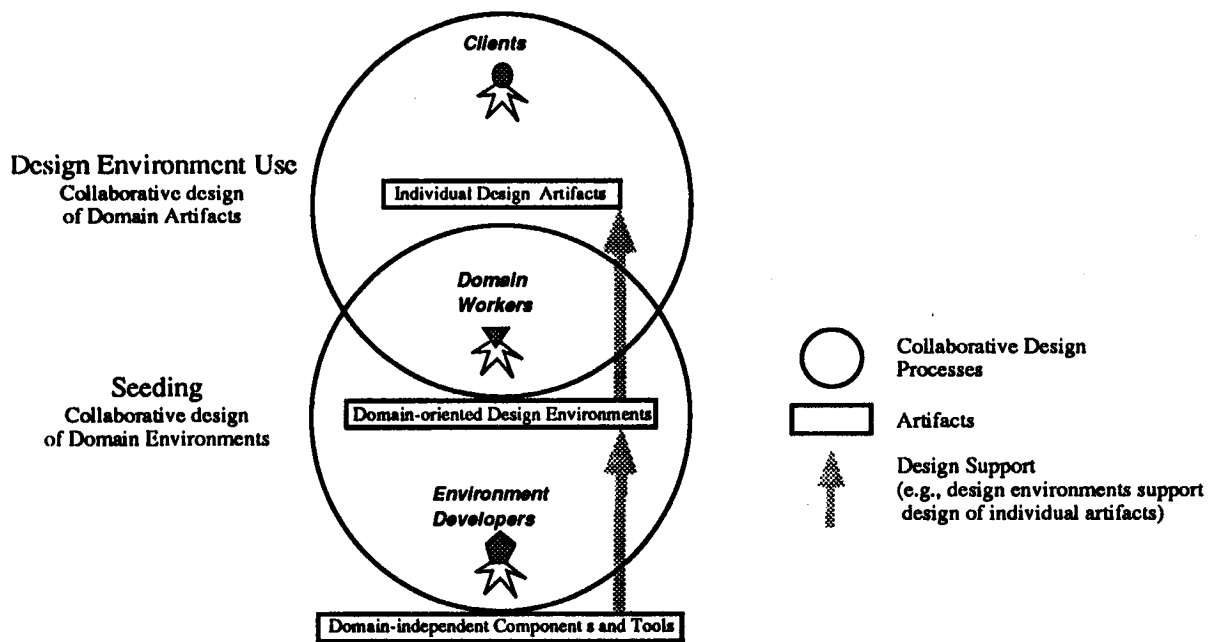


Figure 2: Face to Face Collaborative Interactions

The two ovals illustrate the two levels of direct collaboration involved in building and using domain-oriented design environments. EVA supports design environment builders to produce a design environment seed using domain-independent tools and components in collaboration with domain workers (designers) who will use the design environment. KID supports designers to produce individual design artifacts in collaboration with clients, who will use the individual design artifacts.

Figure 3 describes a process model for collaborative design in developing and using a domain-oriented design environment. The model illustrates the two interrelated indirect collaborative design processes that design environments support: 1) seeding and continual design-in-use (Greenbaum, Kyng, 1991) of *design environments*, and 2) design and ongoing evolution of *individual design artifacts* within the design environment.

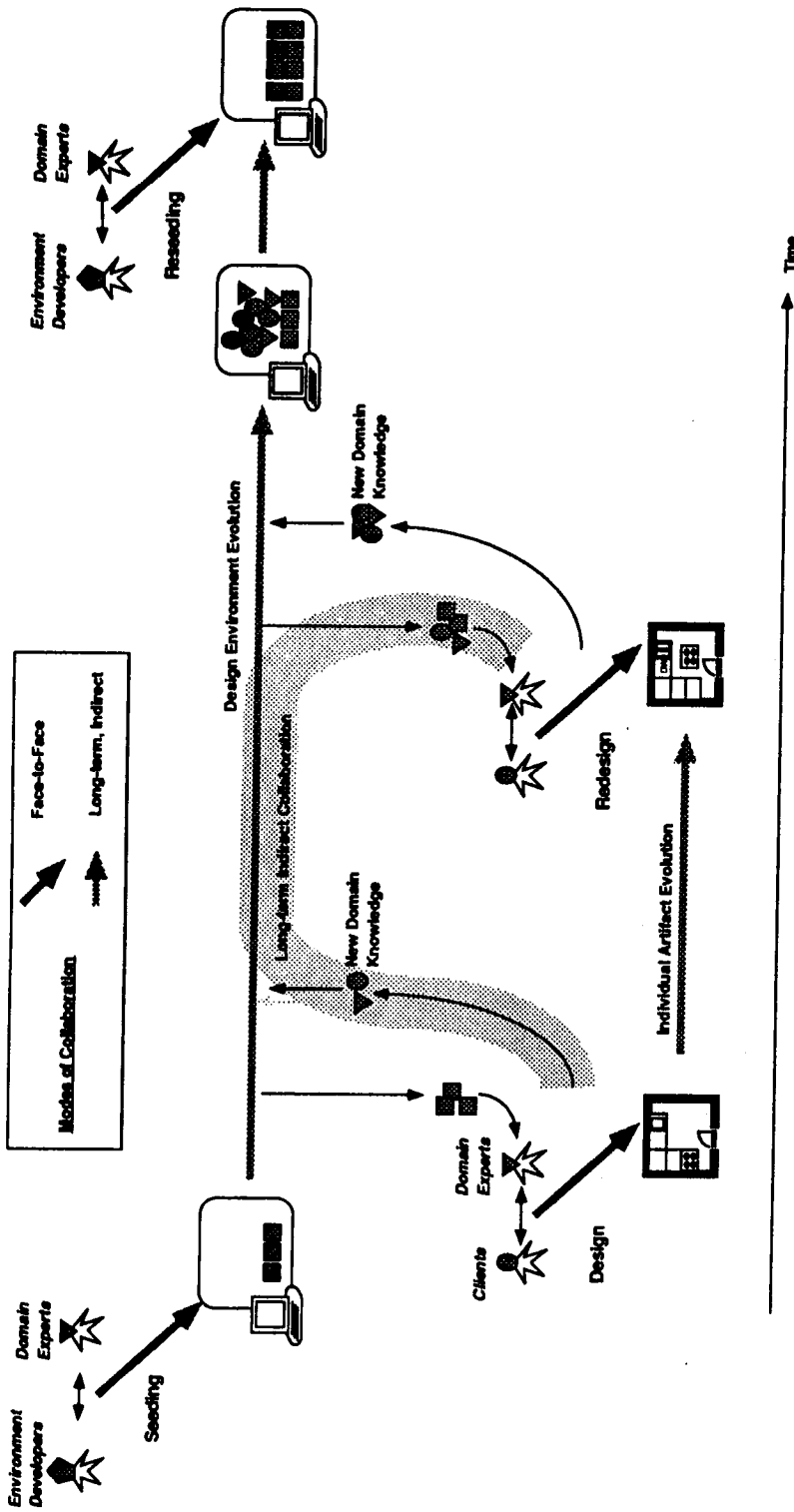


Figure 3: Collaborative Design Processes Facilitated by Design Environments

This figure illustrates major aspects of our work. The top half shows how domain-oriented design environments are created through a collaboration between the environment builders and domain experts. The bottom half shows the use of the design environment in the creation of an individual artifact through a collaboration between domain experts and clients. The use of a design environment contributes to its evolution; i.e., new knowledge, developed in the context of an individual project, is incorporated into the evolving design environment.

3.1 Direct Collaboration Mediated by Design Environments

Seeding. Seed building creates the initial conditions for design environment use. Because design environments are open systems that evolve with use, we refer to the initial system as a seed. The stakeholders are designers (future users of a design environment) and environment builders. The designed artifact is a domain-oriented design environment.

Design. Design is the creation of artifacts. The stakeholders are the designers (the users of the domain-oriented design environment) and the client. The designed artifact is an individual design that is used by the clients.

Both seeding and design processes involve face-to-face collaboration among different stakeholders. Environment builders design domain-oriented design environments in accordance with requirements articulated by designers. The designers in turn design individual artifacts based on clients' requirement specifications. In each collaborative design process, the requirement specification and the solution construction must be integrated (Fischer, Nakakoji, 1992). Therefore, the stakeholders must work together to produce quality artifacts.

Since each stakeholder uses a different representation to express their ideas, goals and intentions, communication breakdowns often take place. For example in kitchen design, client often do not understand a blue-print that a designer provides as a partial design. Explicit representations support the construction of a shared language of doing by providing a common reference for communication.

3.2 Long-term, Indirect Collaboration Through Design Environments

A different type of collaboration is required during the reseeding and the redesign processes. Designers (and design environment builders) have to understand why design decisions have been made in order to reuse or modify it.

Redesign. Artifacts are not designed from scratch but are iteratively refined and components are reused. A complex design artifact can never be complete but is constantly evolving in nature. As illustrated in Figure 3, designers gradually accumulate design knowledge through design practice. Thus, the knowledge-base of the design environment gradually evolves. Ironically, the more information the knowledge base has, the more difficult it is for designers to access the useful information without an adequate support because the information space becomes large and complex.

In order to cope with this information overload problem, design environments provide knowledge delivery mechanisms. By sharing the problem context with designers, design environments make the information space relevant to the task at hand, thereby supporting designers to access design knowledge stored by other designers. Consequently, it enables designers to collaborate with other designers via stored design knowledge.

Reseeding. As designers use the design environment seed to create design artifacts, they may encounter gaps in the knowledge base or discover new knowledge that should be recorded. In both cases, designers can add to the knowledge in the seed. As design knowledge is accumulated over time, the knowledge base will occasionally require *reseeding*. Reseeding is process in which design environment builders reorganize the knowledge-base in order to eliminate inconsistent or obsolete knowledge and to consolidate redundant knowledge. Little research has been done in how to support this reseeding process, but we need to keep in mind that

reseeding should not be considered as a result of flaws in the design of a design environment, but rather be a necessary process for evolution of design environments.

4 Explicit Shared Context and Communication of Intent

Communication between clients and designers is difficult because designers and clients use different languages. Explicit Representations ground collaborative design by providing a context for communication. Representations help to detect communication breakdowns caused by unfamiliar terminology and tacit background assumptions, and turn the breakdowns into opportunities to create a shared understanding. Consequently, explicit representations serve as a basis for the creation of a new and shared language between collaborators.

An important component of shared context is the *intent* of the collaborators. A shared understanding of intent promotes mutual intelligibility by serving as a resource for assessing the relevance of information within the context of collaboration. In everyday communication between people, intent is often implicitly communicated against in the rich background of shared experience and circumstances. Machines, however, have a limited notion of background that limits their ability to infer the intent of users (Suchman, 1987).

Domain-oriented design environments address this problem in three ways. First, a domain-orientation allows a default intent to be assumed, namely, the creation of a “good” artifact in the given domain. Second, a construction situation can be “parsed” by the system, providing the system with information about the artifact under construction. Third, a specification component allows the designer to explicitly communicate high-level design intentions to the system.

In our design environments, design activities, including the communication of intent, are centered around artifacts. By capturing the intentions and priorities of designers and associating them with the artifacts, design environments can have a deep representation of an artifact. This deep representation allows the system to locate stored artifacts and information that are relevant to a designer’s task at hand, and provides the designer with rich resources for assessing the relevance of delivered information.

The first aspect of our work described is the role of a specification and construction components of a design environment during the *use* of the design environment. The specification and construction components of the KID (Knowing-in-Design) environment provide explicit representations of the design intention and partial solution, which ground collaboration among designers and clients. The representations also allow designers to communicate design intentions to the system, thereby establishing a shared context between the designers and the design environment that enables the system to deliver design knowledge relevant to the task at hand. The design knowledge contained in our design environments has been accumulated through previous design efforts, thereby allowing designers to collaborate indirectly with past designers.

The second aspect of our work described is design environment *seeding*, where domain experts and system builders collaboratively design a design environment — a complex artifact in it’s own right. The EVA (Evolving Artifact) approach uses descriptive representations, functional representations, and seed prototypes to facilitate a mutual education process between design environment builders and designers. The evolving seed provides a shared context to ground collaboration between designers having different backgrounds.

5 Collaboration in The KID Design Environment

The KID system is a design environment for creating kitchen floor plans and substantially extends the JANUS system (Fischer, McCall, Morch, 1989). Figure 4 and Figure 5 show screen images of the KIDSPECIFICATION and KIDCONSTRUCTION components of KID. The specification component supports designers in framing their design problem; i.e., specifying design goals, objectives, and criteria or constraints. The construction component support designers in constructing a the solution form of the design artifact. In the kitchen domain the solution form is a floor plan.

The user interface of KIDSPECIFICATION is based on the questionnaire forms used by professional kitchen designers to elicit their clients' requirements. KIDSPECIFICATION provides an extensible collection of questions (issues) and alternative answers from which designers select the requirements associated with their current design task, and assign weights to the selected answers to represent the relative importance of the specified requirements. If no existing alternatives express their position, designers can add or modify information in the underlying argumentation base.

Domain-oriented design environments support a rich notion of design context. In KID the designer determines the context of design by manipulating interface objects in the construction and specification components. The system has access to the context through the state of the interface objects. The specification provides information about the designer's high-level intentions. From the solution construction, the system obtains information about the design moves that have been made. The representations in the specification and construction that define the design context are *shared* between the designers and the system because the state of the representations are accessible to both.

KID uses computational critic mechanisms (Fischer et al., 1993) to alert designers to problematic design situations, such as a violation of domain design rules, and to provide information relevant to the situation. KID contains two collections of domain knowledge: an argumentation base that stores design rationale, and a catalog base that stores design artifacts. The argumentation base is a semi-structured design space that expresses interdependencies between design decisions as well as the contexts in which the interdependencies are relevant. The catalog base contains precedent design cases represented as a construction (floor plan) and a specification (design requirements).

Conceptual coherence in design can be defined as the "match" between problem requirements and problem solution. A fundamental challenge for computational design support is to represent the dependencies between a high-level problem specification and a low-level construction. KID uses *specification-linking rules* to do so. Specification-linking rules map from a preference articulated in the specification to a corresponding combination of constraints that should be satisfied in the construction.

The Specification-linking rules enable KID to detect design situations in which the construction and specification are in conflict. Such conflicts are brought to the designer's attention by *specific critics* (Fischer et al., 1993). Information is provided to help designers understand problematic situations by two knowledge delivery mechanisms:

- **RULE-DELIVERER** locates information in the argumentation base corresponding to the conflict between the specification and construction. The argumentative information helps designers to understand the problem and alternative means for

Specification		Save Current Specification Start New Specification Show Suggestions Quick Question	Load Specifications Copy Specification Set Options Quick Answer	Store Base Issues Catalog Explorer Resume Construction Quick Argument
Catalog MILL-KITCHEN MARK-KITCHEN BILLY-KITCHEN ISAK-KITCHEN JONES-KITCHEN USCAR-KITCHEN PAUL-KITCHEN BRAB-KITCHEN ROB-KITCHEN COLA-KITCHEN CONSER-KITCHEN DENSON-KITCHEN BREYER-KITCHEN GEORGE-KITCHEN CONRALES-KITCHEN	Questions - Entertainment requirement? - Yes - Occasionally - Seldom - Not at all - Methods of Cooking - Do you usually use a microwave? - yes - no - Other Kitchen Activities - Children's Hangout? - yes - no - Do you need a eating space? - yes - no - Preferences - Type of kitchen? - contemporary - traditional - Country - Shape of kitchen? - L-shape - U-shape [answer suggested because hour-many-coo - Corridor - Island [answer suggested because entertainment - One-wall	Current Specifications for: Type: kitchen Name: Joe-kitchen - Size of family? 1 - One - How many cooks usually use the kitchen at once? 2 - one - Is the primary cook right-handed or left-handed? 3 - Left handed - Entertainment requirement? 10 - Yes	Argumentation for Which type of sink do you need? - double bowl sink - Double-Bowl-Sink-Exists(Jc::Whole-Design) (-) If you often entertain, a double bowl sink is preferable. (Dudeall, Kuniya 18-12-91 18:28:18) - single bowl sink - Single-Bowl-Sink-Exists(Jc::Whole-Design) (-) If you have a small family, you may need just a single-bowl sink. (Dudeall, Kuniya	Command - Show My Suggested singl - bowl-sink - Show Arguments type-of-s - Select Issue From Argume - If you often-ent - Toggle Answer yes
Suggested: Issue [43] Which type of sink do you need? - Size of family? One	Answer single bowl If you have a small family, you may need just a single-bowl sink. One	How do I show further argumentation? Buttons: Select this question; House-It; Menu. To save other commands, press Shift, Control, Meta-Shift, or Super- Fri 19 Feb 18:32:54 Kuniya CL SPEC: User Input		

Figure 4: KIDSPECIFICATION

Designers can select answers presented in the *Questions* window. The summary of currently selected answers appears in the *Current Specification* window. Each answer is accompanied with a slider that allows designers to assign a weight representing the relative importance of the answer. *KIDSPECIFICATION* provides an explanation about interdependency between a selected answer and a specific critic fired in *KIDCONSTRUCTION* (Figure 5). The related argument (see the *Argumentation For* window) provides a further explanation about how the specific critic is related to one of the selected answers.

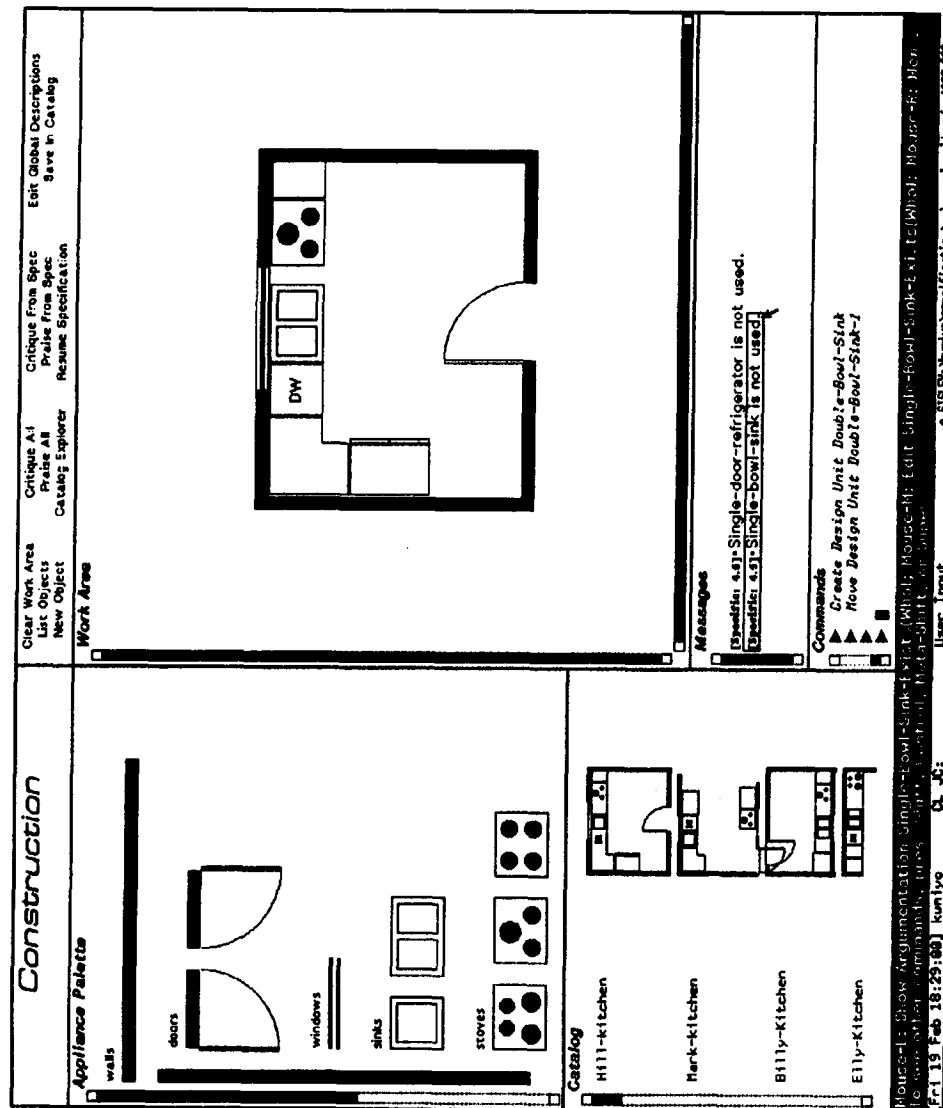


Figure 5: KIDCONSTRUCTION

Designers can construct a kitchen floor plan in *Work Area* by manipulating domain abstractions provided in the *Appliance Palette* window. Designers can also copy and edit a catalog example from the *Catalog* window. The *Messages* window displays a *specific critique message* saying that the current construction in the work area is not compliant to the current specification shown in Figure 4 in terms of types of sink and refrigerator. Catalog examples in the *Catalog* window are automatically ordered in accordance with the current specification (Figure 4). KID provides specific critics (see the *Messages* window), which are associated with numbers for their relative importance. A presented specific critic is linked to the related specification (see the *Suggested* window in Figure 4).

resolving it.

- **CASE-DELIVERER** orders the catalog space so that examples relevant to the current design situation are easily accessible to the designer. **CASE-DELIVERER** computes conformity of each catalog example to the current partial specification by (1) applying specific critics to each catalog example, (2) computing an appropriateness value for the example as the weighted sum of the critic evaluations, (3) ordering the examples according to the values, and (4) presenting the ordered catalog examples.

KID's explicit representations of a problem specification and solution construction facilitates:

- *collaboration between designers and clients*: using the specification component, clients and designers are encouraged to articulate their design problem collaboratively. An explicit representation of problem specification provided by **KIDSPECIFICATION** helps them to achieve and maintain a common understanding of the problem, and prevents them from overlooking important considerations.
- *long-term indirect collaboration*: by having the specification component, the design environment has more shared understanding about the designer's intention for the current task, and thus can deliver task-relevant information. KID's knowledge bases contain design information and artifacts accumulated through past design efforts, enabling designers to collaborate indirectly with their peers from the past.

5.1 Facilitating Direct Collaboration between Designers and Clients

Collaboration between designers and clients is supported by KID through a specification and a construction components. **KIDSPECIFICATION** allows designers to specify their problems in terms of the problem domain. In some domains, **KIDSPECIFICATION** can be used by clients, not only by designers. Expert designers are good at understanding "*languages of the domain*," such as a representation provided in a construction component. Clients, or end-users of application software, often do not understand such languages of the domain.

The specification component of a design environment provides an explicit representation of a problem. While prototypes in the construction component are built by designers, partial problems in the specification component are built by clients.

Specification-linking rules used in KID facilitate the communication between designers and clients by providing a mapping between the construction and the specification. The specification-linking rules represent interdependencies among specification and construction. Clients specify their problem in the specification component and designers build their solution in the construction component. Specification-linking rules can detect inconsistency among the two in the form of specific critics identified by **RULE-DELIVERER** (see the *Messages* window in Figure 5). A relation of the fired specific critics to the current specification is in the the *Suggested* window in Figure 4. KID also helps clients to understand the language that designers use by looking at a concrete representation of a solution (a catalog example) that is retrieved according to the relevance to their problem specification by **CASE-DELIVERER**. The *Catalog* window in Figure 5 lists names of examples that are ordered according to the partial specification.

5.2 Facilitating Indirect Long-term Collaboration

As designers use a design environment to create artifacts, design knowledge is accumulated into the system. A typical example of this type of knowledge includes design rationale and design artifacts. Design rationale explains why certain design decisions are made in what problem situations. It provides heuristics and a source of design expertise in the domain. Design artifacts can be “reused” by designers by combining, or modifying them into a new artifact. They can also be used to warn designers of possible failures that were previously made. Both design rationale together with designed artifacts provide a source for case-based reasoning (Kolodner, 1990; Riesbeck, Schank, 1989).

The argumentation base and the catalog base of design environments facilitate long-term indirect communication among designers. That is, designers communicate through designed artifacts instead of directly talking or writing. Communication is embedded in design artifacts (Reeves, 1993).

As illustrated in Figure 3, a designer accumulates the design knowledge into the system and later another designer can access the design knowledge. A portion of the design rationale articulated and stored by a designer can be used to produce a specification-linking rule. Figure 6 illustrates how the accumulated design rationale can be used to derive specification-linking rules. A detailed description of the mechanism is found in (Nakakoji, 1993). The catalog base can also provide a communication medium among designers. Designers store a design, which helps other designers in producing ideas for a solution. Using catalog examples also amplifies designers’ creativity in performing design (Fischer, Nakakoji, 1993).

KID supports not only recording design experiences but also higher level knowledge acquisition. Competent practitioners usually know more than they can say. This tacit knowledge (Polanyi, 1966) is triggered by new design situations and by breakdowns that occur in a design process. A user study of KID has shown that users of design environments often wanted to extend them in response to breakdowns (Nakakoji, 1993). Thus, as design environments are constantly used, their domain knowledge is increased and refined by interacting with designers.

End-user modifiability of design environments characterizes this aspect. MODIFIER (Girgensohn, 1992) is implemented and tested in the context of JANUS, a precedent of KID. Girgensohn used the representation of objects to allow designers to perform at least some of the adaptations on a level above that of a programming language. For example, using MODIFIER, designers can introduce a new concept such as a microwave oven by copying and modifying attributes of existing concepts such as a stove using a property sheet. Such modification tasks are supported by providing task agendas, on-line help, examples, and critics. Girgensohn identified principles to enhance end-use modifiability of computer systems, including layered architectures and parameterizations.

The approach described here in terms of KID has demonstrated the effectiveness of supporting collaborative design in a relatively mature, stable domain such as kitchen design. For an *immature* or an *unstable* domain, which is relatively new, still under exploration, or heavily depending on state-of-the-art technologies, it is difficult to identify and analyze the domain in consideration. Designers have neither the design knowledge (principles and abstractions), nor ideas of what a solution form should look like.

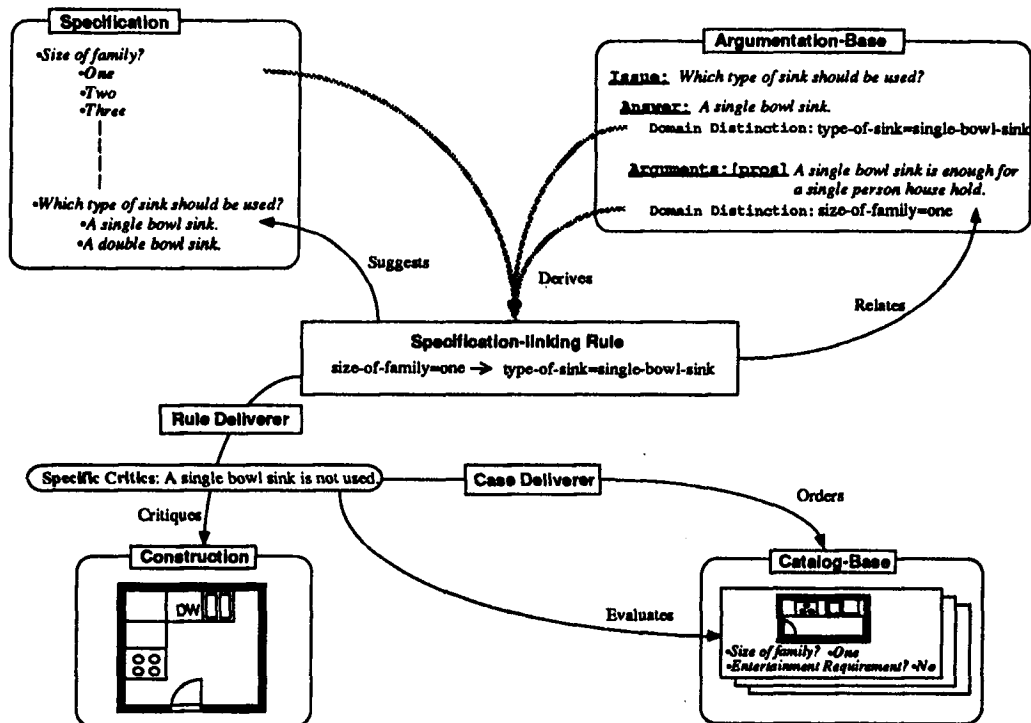


Figure 6: Integration of Components in KID

Specification-linking rules are derived from a partial specification and the argumentation. Derived specification-linking rules are used (1) to make suggestions in KIDSPECIFICATION, (2) to show a related argument in the argumentation base, (3) to identify relevant critics by RULE-DELIVERER, (4) to evaluate a catalog example using the specific critic, and (5) to order the catalog examples by CASE-DELIVERER.

6 The Evolving Artifact Approach to System Building

The kitchen domain has been a useful *object-to-think-with* for our design environment work. The broad-based familiarity and simplicity of the domain makes it ideal for developing and communicating ideas to a wide audience. We have built domain-oriented design environments for other domains such as computer network design (Fischer et al., 1992), lunar habitat design (Stahl, 1993), and phone-based voice dialog design (Repenning, Sumner, 1992). In doing so, we have experienced the difficulty of acquiring the understanding necessary to build design environments for these complex application domains.

The Evolving Artifact (EVA) Approach is a system design methodology to support design environment building in complex domains, where the difficulty is not knowing how to implement system functionality, but rather it is knowing what functionality to implement. The "Evolving Artifact" name refers to the incremental manner in which the design environment is created. EVA approach is a bootstrapping approach, where understanding of the problems to solved guides implementation, and implementation creates understanding. This is in contrast with traditional "waterfall" models of system design, in which a detailed system specification is constructed before implementation proceeds. Waterfall models assume a

complete understanding of the problem exists before implementation begins.

The EVA approach supports system design as a process of mutual education between system builders and domain workers (see Fig2). Because the critical resource in system building is application domain knowledge (Curtis, Krasner, Iscoe, 1988), the initial focus is on representing and understanding application domain concepts. In the EVA approach, *descriptive* representations of application domain concepts provide a context for understanding system requirements, and *functional* representations are used to guide implementation of system functionality. The evolving design environment seed consists of both descriptive objects and functional objects.

In order to further describe the EVA approach, we will use EVA-SERVICE as an example (Ostwald, Burns, Morch, 1992). EVA-service is a design environment to support NYNEX service order representatives to perform the complex task of service provisioning. Service provisioning begins with a service representative and customer collaboratively designing a telephone service configuration to meet the customers requirements. It ends with the implementation of that design, which involves the coordinated activity of many different internal groups, such as line-workers and billing departments. The service representative's job includes entering data in databases, tracking down field personal for progress reports, and providing customers with detailed telephone service information.

6.1 Descriptive and Functional Representations

Descriptive representations. In the initial stages of system design, descriptive representations are the means for facilitating mutual education between system builders and system users. The goal is to build a shared language of design by representing domain concepts in a form that can be discussed, questioned and refined.

Figure 7 shows a screen image containing a descriptive representation of information flow in one perspective of the service provisioning task. The purpose of the diagram is to stimulate discussion that surfaces important issues and terminology about the domain. System builders create descriptive representations and domain workers, who are experts in their domain, point out shortcomings in the diagram or fill in missing knowledge. Modifications are easily made to the descriptive representation, and related concepts and knowledge can be represented and linked using hypermedia technology. For defining and discussing basic concepts, descriptive representations are more useful than conventional object-oriented formalisms.

Functional representations. Descriptive representations are easily understood and modified, but they do not support domain workers to envision how design environments can support their work. In particular, descriptive representations do not have the high degree of interactivity that characterizes most state-of-the-art applications, including design environments.

Functional representations are used to illustrate how computation can be applied to problems and tasks in the application domain. Functional representations are implemented using object-oriented formalisms, and *embedded* in descriptive representations. In the EVA system, function calls may be associated with graphical objects, so that clicking on the object will execute the function. Embedding the functional representations in descriptive representations provides a context for understanding the functionality.

Figure 8 shows a functional representation that is embedded in the descriptive object shown in

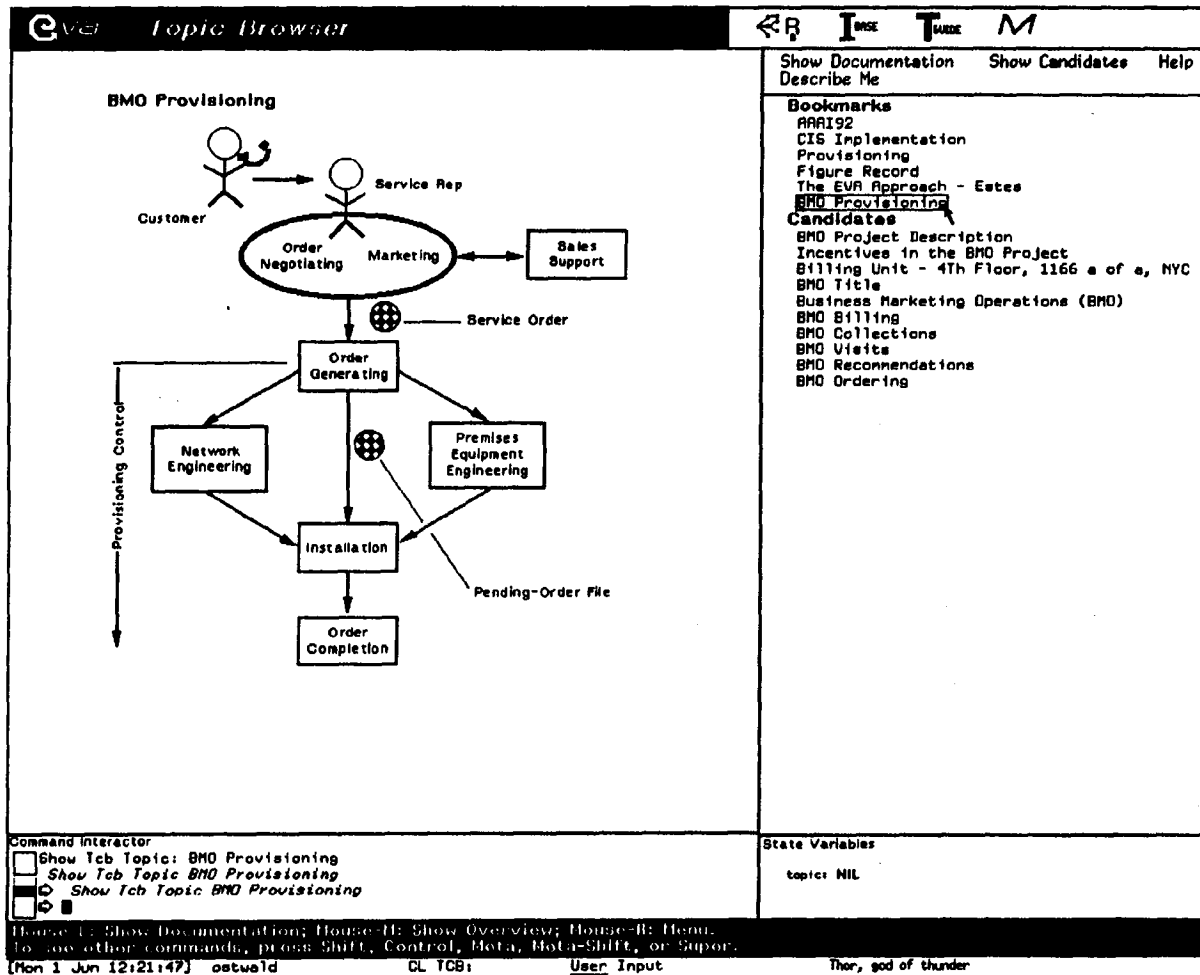


Figure 7: A Descriptive Representation

Descriptive representation as viewed in the "EVA Topic Browser," a tool supporting design environment building. This representation illustrates an aspect of the service provisioning process.

Figure 7. It expresses a possibility of how the task of "order negotiating" might be supported. This functional representation allows service provisioners to evaluate the system builder's understanding of the task, and provides a concrete reference for discussion.

Functional representations are important resources for collaboration, because they help workers to articulate their tacit knowledge. Like the more flexible descriptive representations, functional representations are meant to expose gaps in the collective understanding of the domain and how to support it. Unlike descriptive representations, functional representations allow domain workers to experience functionality in a hands-on manner. This is an important

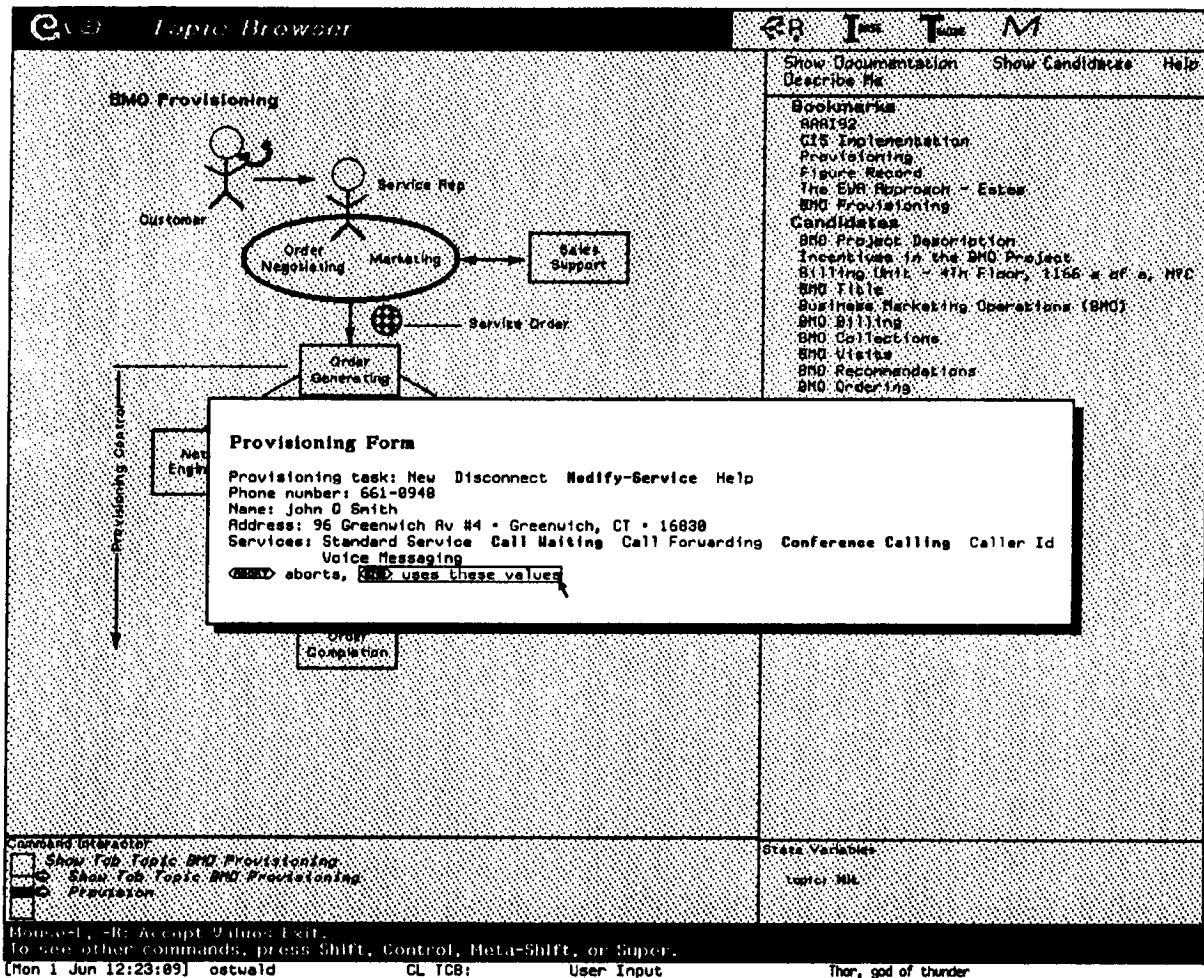


Figure 8: A Functional Representation

A functional representation is embedded in a descriptive object. This form-based representation demonstrates a technique for supporting a data collection task with dynamically configured fields. System builders use the functional representation to elicit knowledge from service provisioners about the interdependencies between fields and field values. Service provisioners use the representation to understand the concept of "smart forms".

part of the mutual education process, because it surfaces tacit knowledge workers have about their work practice, thereby allowing them to contribute their expertise to the task of system building.

Seed Prototypes. Seed prototypes are used to focus mutual education on issues relevant to system use, such as accessing information and collaborating with other users. Seeds are built from descriptive and functional objects, but rather than embedding functionality in descriptive


Provisioning System

Provisioning Form

Provisioning tasks: New Disconnect Modify-Service Help
 Phone number: 661-0948
 Name: bill smith
 Address: 96 Greenwich Av • Greenwich CT • 16883
 Services: Standard Service Call Waiting Call Forwarding Conference Calling Caller Id Voice Messaging
*** aborts, *** uses these values

Call Forwarding

Description



Call forwarding causes a call to one number to be automatically transferred to another.

Sales Presentation

'Call forwarding allows you to get valuable business calls that might otherwise end up on your competitor's desk. . .'
 Special deals for large businesses . . .

Service Requirements

Call forwarding requires both lines (the original line and the line to which the call is forwarded to) to be touch tone. The service is not offered in . . .

Billing Information

MORE

Interact
help Initialize Demo

Help: Select this choice: Help-H: Show documentation; Help-K: Menu; To use other commands, press Shift, Control, Meta/Shift, or Super.
 [Sat 9 May 1:03:41] ostwald CL 6F: User Input Thor, god of thunder

Figure 9: EVA-SERVICE — A Seed Prototype for Service Provisioning

The top portion of the screen is the form-based interface illustrated in Figure 8. Below is information associated with the "call forwarding" field selected with the mouse.

objects, the functional objects are part of the seed interface. In the seed prototype, functional objects are accessed and manipulated in the user interface, while descriptive objects provide information relevant to the user's task at hand. For example, in the KID system describe above, the construction and specification components are functional objects, and the argumentation-base contains descriptive objects.

In the domain of service provisioning, forms play a role analogous to the construction kit of KID. Figure 9 shows a screen image of a EVA-SERVICE prototype based on the functional "form" representation shown in Figure 8. In this screen image, the form is placed in a system framework to simulate how it would appear to users in a fully functional system. Each field of the form provides accesses information that is helpful to the service provisioner. This prototype serves to demonstrate how such information might be accessed in the course of the

service provisioning task. Domain workers will be able to contribute their own information (e.g., sales tips, general information, etc) so that co-workers can access it. This supports indirect collaboration with co-workers.

This prototype does not represent the final product of the seed building process, but instead serves to surface important issues in the collaborative design process. The prototype raises the questions of, “what fields of the provisioning form should lead to descriptive problem-solving information?”, “what types of information are helpful under which circumstances?”, and “how can workers record information so that it doesn’t distract from the task at hand?”

Eventually the evolving EVA-SERVICE prototype will be mature enough to release into the workplace as a design environment for service provisioning domain. As illustrated in Figure 3, EVA-SERVICE will continue to evolve as it is used to support collaborative design between domain workers and clients. Indirect collaboration between domain workers is supported through the accumulation and sharing of service provisioning knowledge.

6.2 Short-term, Direct Collaboration in EVA

The major challenge of system building is to identify and refine the system requirements of domain workers, and to implement appropriate functionality. In the EVA approach representations serve two purposes: (1) they elicit knowledge from domain workers that is often tacit and therefore not expressible in abstract situations, and (2) they communicate the intentions of system builders to domain workers. In both cases, the explicit representations are crucial.

Descriptive representations provide a context for collaboration. They make the intentions of system builders explicit where they can be understood and critiqued by domain workers. Descriptive representations allow the stakeholders to identify and understand important issues early in the design process, before time and effort has been invested in the wrong direction.

Functional representations build upon the understanding gained through creating and modifying descriptive representations. Embedding functionality in descriptive objects provides context in which to understand and discuss the issues raised by applying computation to the application domain. Functional representations support mutual education by allowing system builders to educate domain workers about computational techniques, and by allowing domain workers to educate system builders about what domain knowledge the functionality should contain. Service provisioning workers were able to articulate shortcomings in our functional representations that they did not identify in the corresponding descriptive representations.

Seed prototypes provide a shared context for understanding how functional and descriptive objects should be combined in a mature system. Seed prototypes build upon the shared understanding generated through the construction and evaluation of descriptive and functional objects.

6.3 long-term, Indirect Collaboration in EVA

Long-term, indirect collaboration is required between system builders and system maintainers in domain environment reseeding (see Figure 3). System maintainers need to understand the design decisions that shaped the system. The EVA approach builds an information space of descriptive objects as a natural product of the system building process. The descriptive objects represent the application domain knowledge on which implementation is based — cru-

cial information for reseederers. Because descriptive objects are understandable by domain workers, the objects provide a context for collaboration between system maintainers and domain workers in the reseeding process.

7 Summary

User studies of KID have shown that the explicit representation of client's goals and intentions provided by KIDSPECIFICATION benefits designers in understanding the design problem better by allowing them to (1) create (*frame*) concrete objects to reflect on and (2) reflect on (*see*) a partially framed design with task-relevant design knowledge delivered by KID. Thus, the system enhances the quality of collaboration between designers and clients. The *seeing-framing-seeing* cycle of design processes driven by designers in collaboration with the design environment has been constantly observed in users interacting with KID. Designers' reflection was enhanced by the sometimes *unexpected* arrival of information, which has been stored by other designers. It was observed that the knowledge delivery encouraged designers to articulate design knowledge that had been implicit before. Assessment of these mechanisms illustrates the specification component's beneficial role in enabling the KID design environment to effectively support direct and indirect collaborative design among designers, clients and the design environment.

The EVA approach *extends* our design environment work by applying our design environment work to industrial setting where (1) we cannot choose problems conveniently, and (2) the complexity of the real world is inescapable. The artifacts produced with the EVA approach support long-term, indirect collaboration between system builders by capturing application domain knowledge that is produced during system building. Specifically, descriptive representations that describe important characteristics, terminology and relations of the application domain are useful during the reseeding process. Conventional system building approaches do not focus on *domain* knowledge, and do not integrate this knowledge within the system itself. Consequently, system maintainers are often forced to perform their job without benefit of the knowledge gained by system builders.

Both KID and EVA provide a *process view* of design environments, by greatly facilitating the accumulation of design rationale (Fischer et al., 1991), and by supporting design as a cycle of framing, action, breakdown and repair (Schoen, 1983; Rittel, 1984).

References

- B. Curtis, H. Krasner, N. Iscoe 1988. A Field Study of the Software Design Process for Large Systems, *Communications of the ACM*, 31(11), November:1268-1287.
- 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 Computer-Based Critics in the Contexts of Design, in *Human Factors in Computing Systems, INTERCHI'93 Conference Proceedings*, ACM, (in press).

- G. Fischer, R. McCall, A. Morch 1989. JANUS: Integrating Hypertext with a Knowledge-Based Design Environment, in Proceedings of Hypertext'89 (Pittsburgh, PA), 105-117, ACM, New York.
- 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, K. Nakakoji 1993. Amplifying Designers' Creativity with Domain-Oriented Design Environments, *Artificial Intelligence and Creativity*.
- A. Girgensohn 1992. *End-User Modifiability in Knowledge-Based Design Environments*, Unpublished Ph.D. Dissertation, Department of Computer Science, University of Colorado, Also available as TechReport CU-CS-595-92.
- J. Greenbaum, M. Kyng (eds.) 1991. *Design at Work: Cooperative Design of Computer Systems*, Lawrence Erlbaum Associates, Hillsdale, NJ.
- J.L. Kolodner 1990. *What is Case-Based Reasoning?*, In AAAI'90 Tutorial on Case-Based Reasoning, pp. 1-32.
- 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.
- J. Ostwald, B. Burns, A. Morch 1992. The Evolving Artifact Approach to System Building, in Working Notes of the AAAI 1992 Workshop on Design Rationale Capture and Use, AAAI, 207-214, San Jose, CA.
- M. Polanyi 1966. *The Tacit Dimension*, Doubleday, Garden City, NY.
- B.N. Reeves 1993. *The Role of Embedded Communication and Artifact History in Collaborative Design*, Dissertation Thesis, Department of Computer Science, University of Colorado, Boulder, CO, (forthcoming).
- A. Repenning, T. Sumner 1992. Using Agentsheets to Create a Voice Dialog Design Environment, in Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing, ACM Press, 1199-1207, (also published as Technical Report CU-CS-576-92).
- C. Riesbeck, R.C. Schank 1989. *Inside Case-Based Reasoning*, Lawrence Erlbaum Associates, Hillsdale, NJ.
- H.W.J. Rittel 1984. *Second-Generation Design Methods*, in N. Cross (ed.), *Developments in Design Methodology*, John Wiley & Sons, New York, 317-327.
- D.A. Schoen 1983. *The Reflective Practitioner: How Professionals Think in Action*, Basic Books, New York.
- G. Stahl 1993. *Interpretation in Design: The Problem of Tacit and Explicit Understanding in Computer Support of Cooperative Design*, Unpublished Ph.D. Dissertation, Department of Computer Science, University of Colorado, Forthcoming.
- L.A. Suchman 1987. *Plans and Situated Actions*, Cambridge University Press, Cambridge, UK.