University of Colorado at Boulder

Department of Computer Science

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

EMPOWERING DESIGNERS WITH INTEGRATED DESIGN ENVIRONMENTS

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

[1]Department of Computer Science and Institute of Cognitive Science
University of Colorado
Boulder, Colorado 80309, USA

[2]Software Engineering Laboratory
Software Research Associates, Inc.
1113 Spruce Street, Boulder, Colorado 80302, USA

E-mail: gerhard@cs.colorado.edu; kumiyo@cs.colorado.edu

IN PROCEEDINGS OF THE FIRST INTERNATIONAL CONFERENCE ON AI IN DESIGN, EDINBURGH, UK, JUNE 1991

Abstract: Designers deal with ill-defined and wicked problems characterized by fluctuating and conflicting requirements. Traditional design methodologies based on the separation between problem setting (analysis) and problem solving (synthesis) are inadequate to solve these problems. These types of problems require a cooperative problem-solving approach empowering designers with integrated, domain-oriented, knowledge-based design environments.

In this paper, we describe the motivation for this approach and introduce an architecture for such design environments. We focus on the integration of specification, construction, and a catalog of prestored design objects in those environments for an illustration of how such integrated environments empower human designers. The system component described in detail (called CATALOGEXPLORER) assists designers in locating examples in the catalog that are relevant to the task at hand as partially articulated by the current specification and construction, thereby relieving users of the task of forming queries for retrieval.

Acknowledgements: The authors would like to thank the members of the Human-Computer Communication group at the University of Colorado, who contributed to the conceptual framework and the systems discussed in this paper. The research was supported by Software Research Associates, Inc. (Tokyo, Japan), by the National Science Foundation under grants No. IRI-8722792 and IRI-9015441, and by the Army Research Institute under grant No. MDA903-86-C0143.

EMPOWERING DESIGNERS WITH INTEGRATED DESIGN ENVIRONMENTS

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

[1]Department of Computer Science and Institute of Cognitive Science
University of Colorado
Boulder, Colorado 80309, USA

[2]Software Engineering Laboratory
Software Research Associates, Inc.
1113 Spruce Street, Boulder, Colorado 80302, USA

Abstract. Designers deal with ill-defined and wicked problems characterized by fluctuating and conflicting requirements. Traditional design methodologies based on the separation between problem setting (analysis) and problem solving (synthesis) are inadequate to solve these problems. These types of problems require a cooperative problem-solving approach empowering designers with integrated, domain-oriented, knowledge-based design environments.

In this paper, we describe the motivation for this approach and introduce an architecture for such design environments. We focus on the integration of specification, construction, and a catalog of prestored design objects in those environments for an illustration of how such integrated environments empower human designers. The system component described in detail (called CATALOGEXPLORER) assists designers in locating examples in the catalog that are relevant to the task at hand as partially articulated by the current specification and construction, thereby relieving users of the task of forming queries for retrieval.

INTRODUCTION

Design is an ill-defined (Simon, 1973) or wicked (Rittel, 1984) problem with fluctuating and conflicting requirements. Early design methods, based on directionality, causality, and separation of analysis from synthesis, are inadequate to solve such problems (Cross, 1984).

The research effort discussed in this paper is based on the assumption that these design problems are best solved by supporting a cooperative problem-solving approach between humans and integrated, domain-oriented, knowledge-based design environments (Fischer, 1990). Combining knowledge-based systems and innovative human-computer communication techniques empowers humans to produce "better" products by augmenting their intellectual capabilities and productivity rather than simply by using an automated system that may not be entirely appropriate (Stefik, 1986).

Our approach is not to build another expert system. Expert systems require a rather complete understanding of a problem to start with — an assumption that does not hold for ill-defined problems. In order to produce a set of rules for an expert system, the relevant factors and the background knowledge need to be identified. However, we cannot fully articulate this information. What has been made explicit always sets a limit, and there exists the potential of breakdowns that call for moving beyond this limit (Winograd and Flores, 1986).

In this paper, we will use the domain of architectural design of kitchen floor plans as an ''object-to-think-with'' for purposes of illustration. The simplicity of the domain helps in concentrating on the essential issues of our approach without being distracted by understanding the semantics of the domain itself. We first discuss issues with design environments and emphasize the importance of domain orientation and integration of those environments. Then we describe integrated, domain-oriented, knowledge-based design environments based on the multifaceted architecture as a theoretical framework. Next, an innovative system component, CATALOGEXPLORER, is described as an illustration of how such an integrated environment empowers human designers. The system integrates specification, construction, and a catalog of prestored design objects. The synergy of integration enables the system to retrieve design objects that are relevant to the task at hand as articulated by a partial specification and construction, thereby relieving users of the task of forming queries for retrieval. We discuss related work and conclude with a discussion of achievements, limitations, and future directions.

PROBLEMS

Integration of problem setting and problem solving

Integration of problem setting and problem solving is indispensable (Schoen, 1983). As Simon (1981) mentioned, complex designs are implemented over a long period of time and are continually modified during the whole design process. Simon stated that they have much in common with painting in oil, where current goals lead to new applications of paint, while the gradually changing pattern suggests new goals. One cannot gather information meaningfully unless one has understood the problem, and one cannot understand the problem without information about it. Professional practitioners have at least as much to do with defining the problem as with solving the problem (Rittel, 1984).

An empirical study by our research group, which analyzed *human-human cooperative problem solving* between customers and sales agents in a large hardware store (Reeves, 1990), provided ample evidence that in many cases humans are initially unable to articulate complete requirements for ill-defined problems. Humans start from a partial specification and refine it incrementally, based on the feedback they get from their environment.

The integration of problem setting (analysis) and problem solving (synthesis) is not supported in first-generation design methodologies or in traditional approaches of software design (Sheil, 1983). Automated design methodologies fail because they assume that complete requirement specification can be established before starting design.

Retrieval of information relevant to the task at hand

In supporting integration of problem setting and problem solving in design environments, supporting retrieval of information relevant to the task at hand is crucial. Every step made by a designer toward a solution determines a new space of related information, which cannot be determined a priori due to its very nature. Integrated design environments are based on high-functionality systems (Lemke, 1989) containing a large number of design objects.

High-functionality systems increase the likelihood that an object exists that is close to what is needed — but without adequate system support it is difficult to locate and understand the objects relevant to the task at hand (Figure 1) (Nielsen and Richards, 1989; Fischer and Girgensohn, 1990).

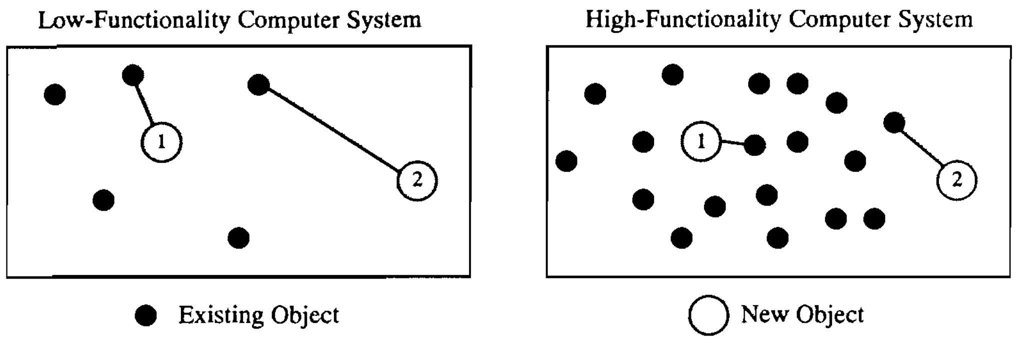

Low-Functionality Computer System
High-Functionality Computer System
1
2
1
2
Existing Object
New Object

Figure 1: Trade-off Between Accessibility and Usefulness

It is easier to locate existing objects in a low-functionality computer system, but the potential for finding an object closer to what is needed is higher in a high-functionality system. The length of lines represents the distance between desired objects and existing objects.

The task at hand cannot be articulated at the beginning of a design, leading to the inapplicability of conventional information retrieval techniques (Fischer, Henninger, and Redmiles, 1991). In a conventional *query-based search*, a highly specific query has to be formulated. If users can articulate what they need, a query-based search takes away a lot of the burden of locating promising objects (Henninger, 1990).

In *navigational access* provided by a browsing mechanism, users tend to get lost while wandering around in the space looking for some target information if the space is large and the structure is complex (Halasz, 1988). Navigational access requires that the information space has a fairly rigid and predetermined structure, making it impossible to tailor the structure according to the task at hand. Browsing mechanisms become useful once the space is narrowed by identifying a small set of relevant information.

Design environments need additional other mechanisms (as discussed in this paper) that can identify small sets of objects relevant to the task at hand. Users must be able to incrementally articulate the task at hand. The information provided in response to these problem-solving activities based on partial specifications and constructions must assist users in refining the definition of their problem.

Domain orientation

To reduce the great transformation distance between a design substrate and an application domain (Hutchins, Hollan, and Norman, 1986), designers should perceive design as communication with an application domain. The computer should become invisible by supporting ***human problem-domain communication***, not just human-computer communication (Fischer and Lemke, 1988). Human problem-domain communication provides a new level of quality in human-computer communication by building the important abstract operations and objects in a given area directly into a computer-supported environment. Such an environment allows designers to design artifacts from application-oriented building blocks of various levels of abstractions, according to the principles of the domain.

Integrated design environments

Design should be an ongoing process of cycles of specification, construction, evaluation, and reuse in the working context. At each stage in the design process, the partial design embedded in the design environment serves as a stimulus for suggesting what users should attend to next. This direction to new subgoals permits new information to be extracted from memory and reference sources and another step to be taken toward the development of the design. Thus, the integration of various aspects of design enables the situation to ''*talk back*'' to users (Schoen, 1983) by providing them with immediate and clear feedback of the current problem context.

By virtue of the synergy of integration, such environments can partially articulate the user's task at hand by a partial specification and construction. As a consequence, the users can be provided with the information relevant to the task at hand by the system without forming queries for the retrieval. The use of the information is also supported in the same environment; thereby the system can analyze usage patterns of the retrieved information and use them for refining the retrieval.

A MULTIFACETED ARCHITECTURE FOR INTEGRATED DESIGN ENVIRONMENTS

During the last five years, we have developed and evaluated several prototype systems of domain-oriented design environments (Fischer, McCall, and Morch, 1989; Lemke and Fischer, 1990). Different system-building efforts led to the multifaceted architecture that will be described in the context of the JANUS system. The domain of JANUS is the architectural floor plan design of a kitchen. The system is implemented in Common Lisp, and runs on Symbolics Lisp machines. Currently JANUS consists of subsystems JANUS-CONSTRUCTION, JANUS-ARGUMENTATION, and CATALOGEXPLORER. Each subsystem supports different aspects of the architecture.

Although we have emphasized the importance of domain orientation, this architecture should not be regarded as a specific framework for a certain domain. To the contrary, we assume that the architecture presented here serves as a generic framework for constructing a class of domain-specific environments.

Components of the multifaceted architecture

The multifaceted architecture for integrated design environments consists of the following five components (Figure 2).

- A **construction kit** is the principal medium for implementing design. It provides a palette of domain abstractions and supports the construction of artifacts using direct manipulation and other interaction styles. A construction represents a concrete implementation of a design and reflects a user's current problem situation. Figure 3 shows the screen image of JANUS-CONSTRUCTION, which supports this role.

- An **issue-based argumentative hypermedia system** captures the design rationale. Information fragments in the hypermedia issue base are based on an issue-based information system (IBIS; McCall, 1986), and are linked according to what information serves to resolve an issue relevant to a partial construction. The issues, answers, and arguments held in JANUS-ARGUMENTATION (see Figure 4) can be accessed via links from the domain knowledge in other components.

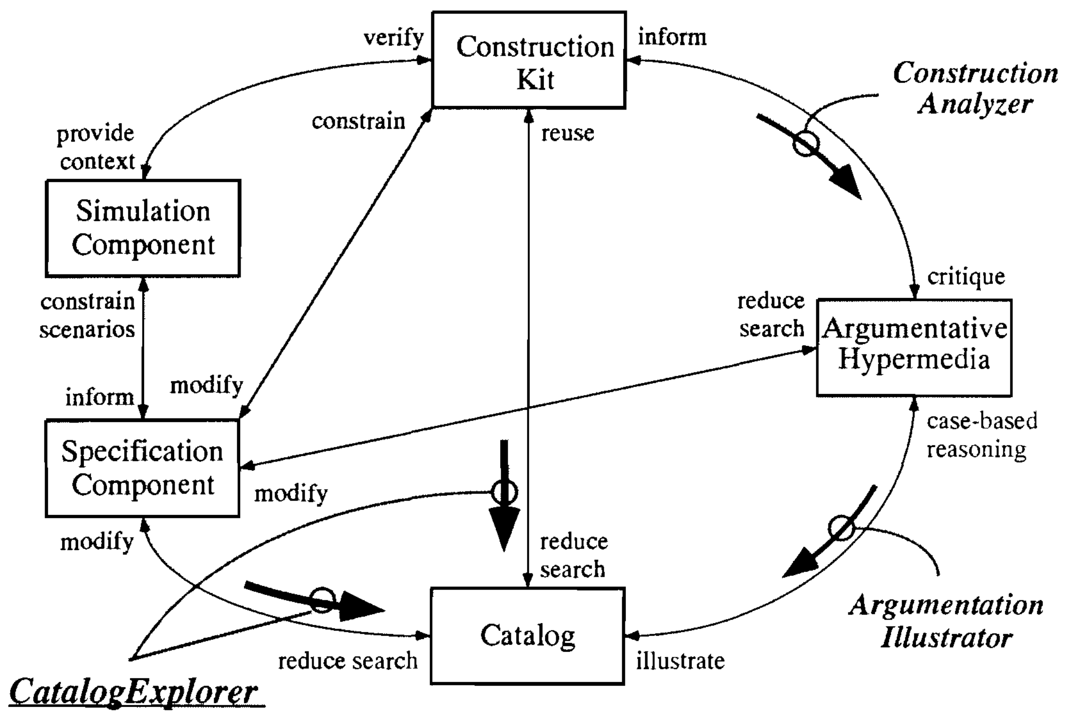


Figure 2: A Multifaceted Architecture

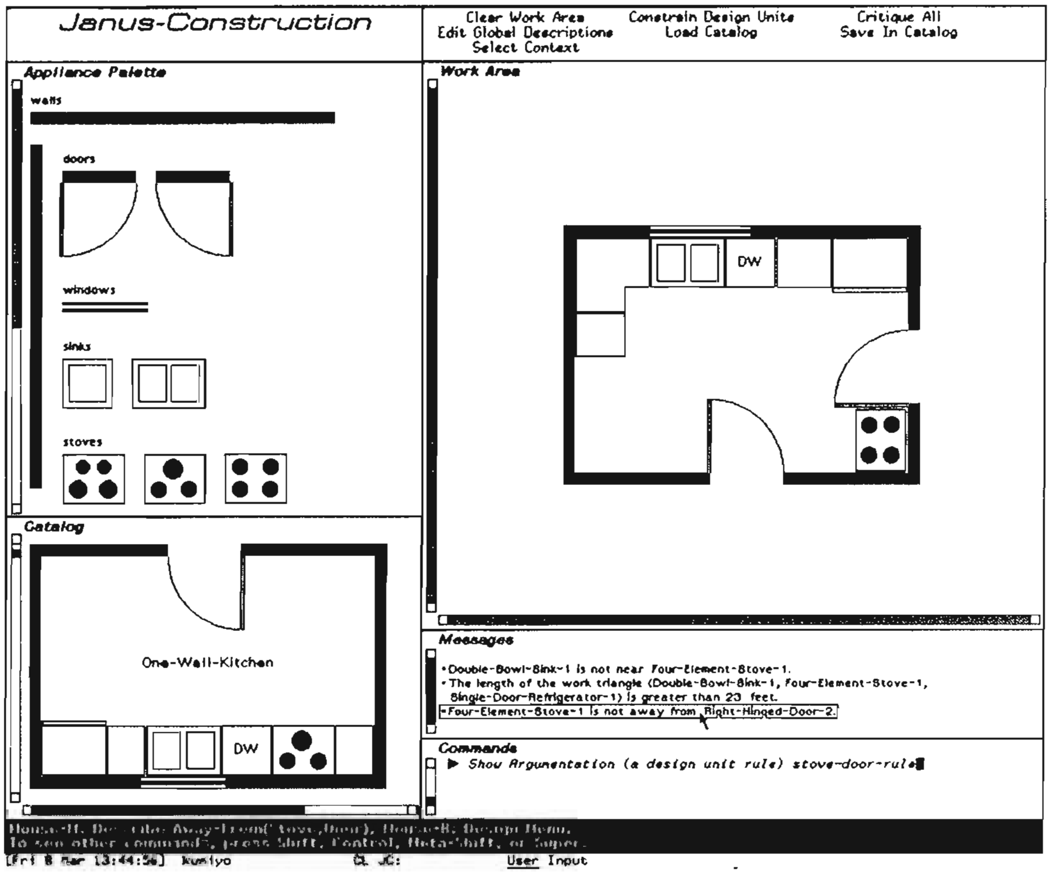
The components of the multifaceted architecture for an integrated design environment. Support for links between the components are crucial for synergy of integration.

- A **catalog** (see Figures 3 and 6) provides a collection of prestored design objects illustrating the space of possible designs in the domain. Catalog examples support reuse and case-based reasoning (Kolodner, 1990; Riesbeck and Schank, 1989).
- A **specification component** (see Figure 7) allows designers to describe some characteristics of the design they have in mind. The specifications are expected to be modified and augmented during the whole design process, rather than to be fully articulated before starting the design. CATALOGEXPLORER provides this mechanism. After specification, users are asked to weigh the importance of each item (Figure 8).
- A **simulation component** allows one to carry out "what-if" games to let designers simulate usage scenarios with the artifact being designed. Simulation complements the argumentative component.

Links among the components

The architecture derives its essential value from the integration of its components and links between the components. Used individually, the components cannot achieve their full potential. Used in combination, however, each component augments the value of the others, forming a synergistic whole.

Links among the components of the architecture are supported by various mechanisms (see Figure 2). The integration enables the system to incrementally understand the task at



•Double-Bowl-Sink-1 is not near Four-Element-Stove-1.
•The length of the work triangle (Double-Bowl-Sink-1, Four-Element-Stove-1, Single-Door-Refrigerator-1) is greater than 23 feet.
•Four-Element-Stove-1 is not away from Right-Hinged-Door-2.

```
▶ Show Argumentation (a design unit rule) stove-door-rule
```

Figure 3: Screen Image of JANUS-CONSTRUCTION

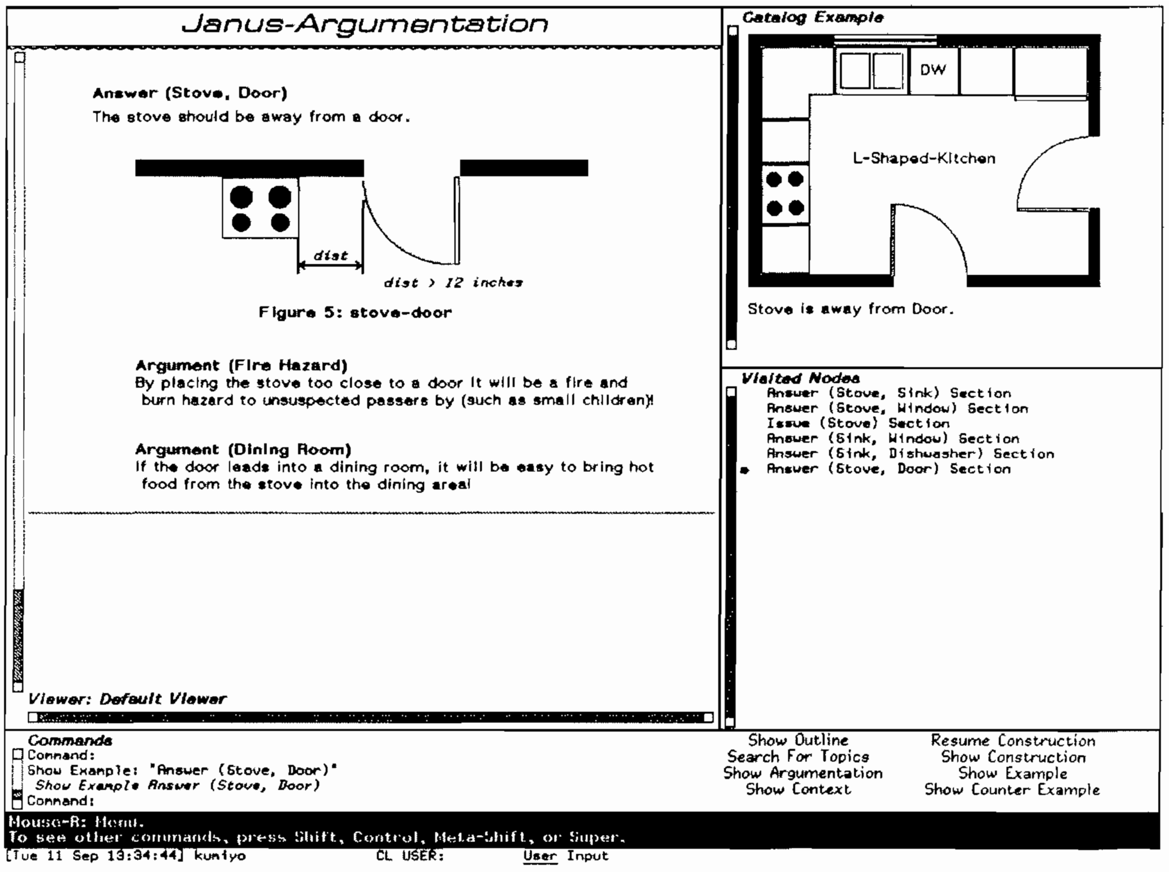
This screen image shows JANUS-CONSTRUCTION, the construction component of JANUS. Building blocks (design units) are selected from the *Palette* and moved to desired locations inside the *Work Area*. Designers can reuse and redesign complete floor plans from the *Catalog*. The *Messages* pane displays critiques automatically after each design change that triggers such a critic message (done by CONSTRUCTION ANALYZER). Clicking with the mouse on a message activates JANUS-ARGUMENTATION and displays the argumentation related to that message (see Figure 4).

hand, thereby providing users with the information relevant to the task at hand. The major mechanisms to achieve this are:

- *CONSTRUCTION ANALYZER* is a critiquing component (Fischer et al., 1990) that detects and critiques partial solutions constructed by users based on domain knowledge of design principles. The firing of a critic signals a breakdown to designers (Winograd and Flores, 1986), warning them of potential problems in the current construction, and providing them with an immediate entry into the exact place in the argumentative hypermedia system where the corresponding argumentation lies (see Figures 3 and 4).

- *ARGUMENTATION ILLUSTRATOR* helps users to understand the information given in an argumentative hypermedia by using a catalog design example as a source of concrete realization (see Figure 4). The explanation given as an argumentation is often highly abstract and very conceptual. Concrete design examples that match the explanation help users to understand the concept.

- *CATALOGEXPLORER*, described later in detail, helps users to search the catalog space



Answer (Stove, Door)
The stove should be away from a door.

Figure 5: stove-door

Argument (Fire Hazard)
By placing the stove too close to a door it will be a fire and burn hazard to unsuspected passers by (such as small children)!

Argument (Dining Room)
If the door leads into a dining room, it will be easy to bring hot food from the stove into the dining area!

Viewer: Default Viewer

Stove is away from Door.

```
Visited Nodes
Answer (Stove, Sink) Section
Answer (Stove, Window) Section
Issue (Stove) Section
Answer (Sink, Window) Section
Answer (Sink, Dishwasher) Section
Answer (Stove, Door) Section
```

```
Commands
Command:
Show Example: "Answer (Stove, Door)"
 Show Example Answer (Stove, Door)
Command:
Mouse-R: Menu.
To see other commands, press Shift, Control, Meta-Shift, or Super.
[Tue 11 Sep 13:34:44] kuniyo          CL USER:          User Input
```

Figure 4: Screen Image of JANUS-ARGUMENTATION

This screen image of JANUS-ARGUMENTATION shows an answer to the issue of where to locate the kitchen stove with respect to a door, and graphically indicates the desirable relative positions of the two design units. Below this is a list of arguments for and against the answer. The example in the upper right corner (activated by the "Show Example" command in the *Commands* pane) contextualizes an argumentative principle in relation to a specific design (done by ARGUMENTATION ILLUSTRATOR).

according to the task at hand. It retrieves design examples similar to the current construction situation and orders a set of design examples by their appropriateness to the current specification.

Design within the multifaceted architecture

Figure 5 illustrates the coevolution of specification and construction in an environment based on the multifaceted architecture. A typical cycle of events in the environments includes: (1) designers create a partial specification or a partial construction; (2) they do not know how to continue with this process; (3) they switch and consult other components in the system, being provided with information relevant to the partially articulated task at hand; and (4) they are able to refine their understanding based on the *back talk* of the situation. As designers go back and forth among these components, the problem space is narrowed and all facets of the artifact are refined. A completed design artifact consisting of specification and construction may be stored into the catalog for later reuse. Thus, the environment gradually evolves itself by being continually used.

Problem analysis and synthesis are thus integrated in such an environment, following Schoen's (1983) characterization of design activities:

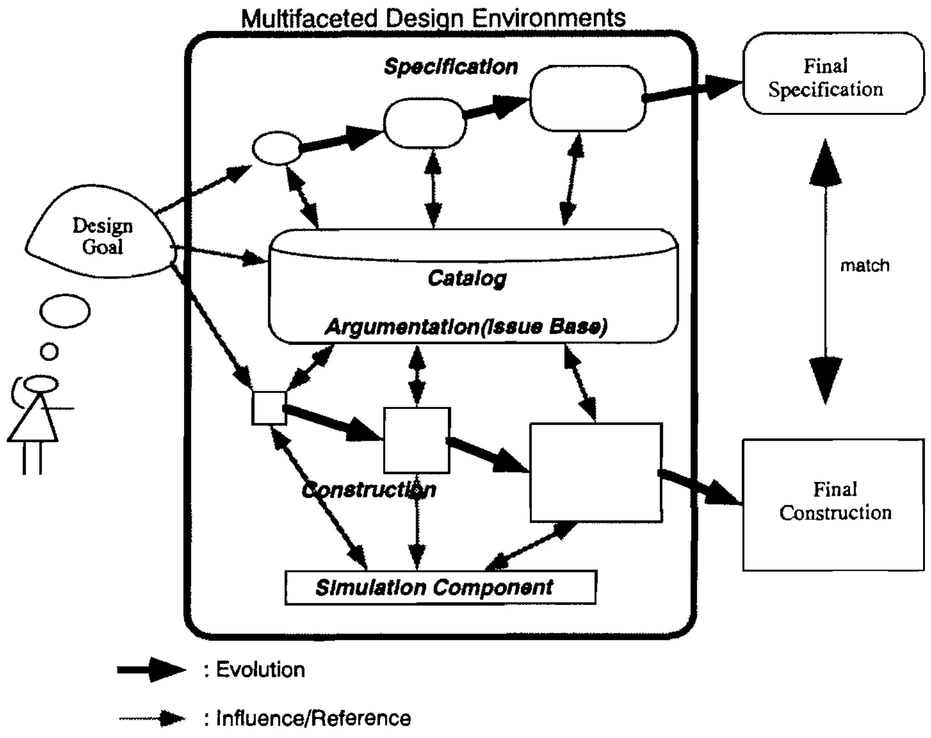


Figure 5: Coevolution of Construction and Specification of Design in Multifaceted Architecture

Starting with a vague design goal, designers go back and forth among the components in the environment. During the process, a designer and the system cooperatively evolve a specification and a construction incrementally, by utilizing the available information in an argumentation component and a catalog and feedback from a simulation component. In the end, the outcome is a matching pair of specification and construction.

Sometimes modification of a specification leads a designer directly to modify a construction, or vice versa. Instead of evolving them, a designer may replace the current construction or specification by reusable design objects. A cycle ends when a designer commits the completion of the development.

> The designer shapes the situation in accordance with his initial appreciation of it [construction], the situation "talks back" [critics], and he responds to the situation's back-talk. In a good process of design, this conversation with the situation is reflective. In answer to the situation's back-talk, the designer reflects-in-action on the construction of the problem [argumentation].

Schoen's work provides interesting insights into design processes, but it does not provide any mechanisms to support the approach. Our system-building efforts (McCall, Fischer, and Morch, 1990; Fischer, 1989) are oriented toward the goal of creating these support mechanisms for the theory.

CATALOGEXPLORER

In this section, we describe CATALOGEXPLORER, which links the specification and construction components with the catalog (see Figure 2), followed by a scenario that illustrates a typical use of the system. In the following two sections, we describe the underlying mechanisms used in the scenario in more detail, including the mechanisms of retrieval from specification and retrieval from construction, respectively.

System description

Design objects stored in a catalog can be used for (1) providing a solution to a new problem, (2) warning of possible failures, and (3) evaluating and justifying the decision (Kolodner, 1990; Rissland and Skalak, 1989). The catalog provides a source for different ideas such as commercial catalogs shown by a professional kitchen designer to customers to help them understand their needs and make decisions. For large catalogs, identifying design examples relevant to the task at hand becomes a challenging and time-consuming task.

By integrating specification, construction, and a catalog, CATALOGEXPLORER helps users to retrieve information relevant to the task at hand and, as a result, helps users to refine their partial specification and partial construction. Users need not form queries for retrieving design objects from a catalog because their task at hand is partially articulated by a partial specification and construction.

The design examples in the catalog are stored as objects in a knowledge base. Each design example consists of a floor layout and a set of slot values. The examples are automatically classified according to their explicitly specified features, the slot values provided by a user. Each design example can be (1) critiqued and praised by CONSTRUCTION ANALYZER, and (2) marked with a bookmark, which provides users with control in selecting design examples and forming a personalized small subset of the catalog.

CATALOGEXPLORER is based on the HELGON system (Fischer and Nieper-Lemke, 1989), which instantiates the retrieval by reformulation paradigm (Williams, 1984). It allows users to incrementally improve a query by critiquing the results of previous queries. Reformulation allows users to iteratively search for more appropriate design information and to refine their specification, rather than being constrained to their specified query in the first place (Fischer, Henninger, and Redmiles, 1991).

Based on the retrieval by reformulation paradigm, CATALOGEXPLORER retrieves design objects relevant to the task at hand by using the following mechanisms:

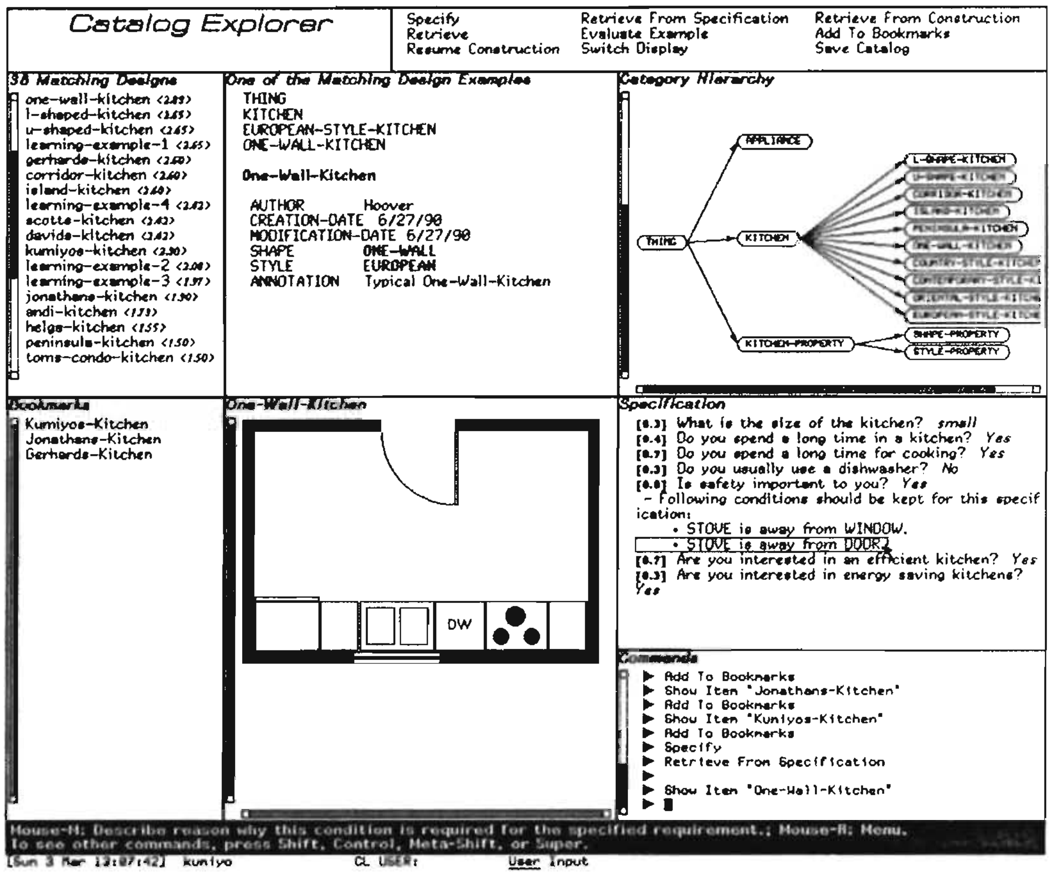
- It provides a specification sheet and a mechanism to differentiate the importance of each specification item by assigning weights to them. It orders design examples by computed appropriateness values based on the specification.

- It analyzes the current construction and retrieves similar examples from the catalog.

A scenario using CATALOGEXPLORER

CATALOGEXPLORER (Figure 6) is invoked by the *Catalog* command from JANUS-CONSTRUCTION (Figure 3). The *Specify* command provides a *specification sheet* (Figure 7) in the form of a questionnaire. After specification, users are asked to assign a weight to each specified item in a *weighting sheet* (Figure 8).

The specified items are shown in the *Specification* window in Figure 6. By clicking on one of the specified items, users are provided with physical necessary condition rules (*specification-linking rules*) for a kitchen design to satisfy the specified item, as seen in the two lines in the middle of the *Specification* window in Figure 6. With this information, users can explore the arguments behind the rules. The shown condition rules are mouse-sensitive, and clicking on one of them will activate JANUS-ARGUMENTATION providing more detailed information. Figure 4 illustrates the rationale behind the rule "*the stove*



38 Matching Designs
one-wall-kitchen <2.89>
l-shaped-kitchen <2.85>
u-shaped-kitchen <2.65>
learning-example-1 <2.65>
gerhards-kitchen <2.50>
corridor-kitchen <2.60>
island-kitchen <2.60>
learning-example-4 <2.42>
scotts-kitchen <2.42>
davids-kitchen <2.42>
kumiyos-kitchen <2.30>
learning-example-2 <2.08>
learning-example-3 <1.97>
jonathans-kitchen <1.90>
andi-kitchen <1.73>
helgas-kitchen <1.55>
peninsula-kitchen <1.50>
toms-condo-kitchen <1.50>

One of the Matching Design Examples
THING
KITCHEN
EUROPEAN-STYLE-KITCHEN
ONE-WALL-KITCHEN

One-Wall-Kitchen

AUTHOR	Hoover
CREATION-DATE	6/27/90
MODIFICATION-DATE	6/27/90
SHAPE	ONE-WALL
STYLE	EUROPEAN
ANNOTATION	Typical One-Wall-Kitchen

Bookmarks
Kumiyos-Kitchen
Jonathans-Kitchen
Gerhards-Kitchen

One-Wall-Kitchen

Specification
[0.3] What is the size of the kitchen? *small*
[0.4] Do you spend a long time in a kitchen? *Yes*
[0.7] Do you spend a long time for cooking? *Yes*
[0.3] Do you usually use a dishwasher? *No*
[0.8] Is safety important to you? *Yes*
 - Following conditions should be kept for this specification:
   • STOVE is away from WINDOW.
   • STOVE is away from DOOR.
[0.7] Are you interested in an efficient kitchen? *Yes*
[0.3] Are you interested in energy saving kitchens? *Yes*

Commands
```
▶ Add To Bookmarks
▶ Show Item "Jonathans-Kitchen"
▶ Add To Bookmarks
▶ Show Item "Kumiyos-Kitchen"
▶ Add To Bookmarks
▶ Specify
▶ Retrieve From Specification
▶
▶ Show Item "One-Wall-Kitchen"
▶ █
```

```
Mouse-M: Describe reason why this condition is required for the specified requirement.; Mouse-R: Menu.
To see other commands, press Shift, Control, Meta-Shift, or Super.
[Sun 3 Mar 13:07:42]   kumiyo            CL USER:            User Input
```

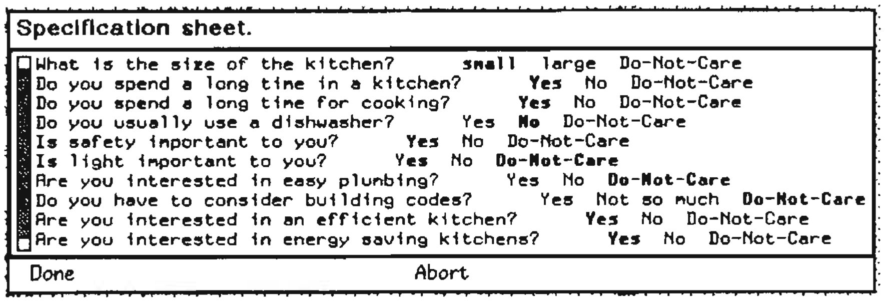
Figure 6: Screen Image of CATALOGEXPLORER

The leftmost *Matching Designs* window lists the names of all matching design examples in the catalog. The numbers following the names represent the *appropriateness* values of each design. The *Bookmarks* window stores some of the previously visited catalog items. The two panes in the middle show one of the matching examples in detail (the top pane shows a set of slot values and the bottom pane a floor layout). The *Category Hierarchy* window shows the hierarchical structure of the catalog. The *Specification* window shows the specified items with assigned weights (see Figures 7 and 8).

should be away from a door if a user wants a kitchen to be safe.'' By invoking the *Retrieve From Specification* command, the design examples of the catalog are ordered (see the *Matching Designs* window in Figure 6) by *appropriateness* values to the specified items.

Users can then retrieve design examples similar to their current construction. When invoking the *Retrieve From Construction* command, users are asked to choose a criterion (*parsing topic*) for defining the similarity between the current construction and design examples in the catalog. When users choose ``*Design Unit Types*'' as a parsing topic, a menu comes up as shown in Figure 9, allowing the user to select all or some of the design unit types being used in the current construction. In Figure 9, a user has selected all appliances that were used in the construction of Figure 3. The system then retrieves examples that contain the specified design unit types.

The above interactions gradually narrow the catalog space, providing users with a small set of examples relevant to the current construction and ordered by the appropriateness to their specification. Users can examine them one by one with a reasonable amount



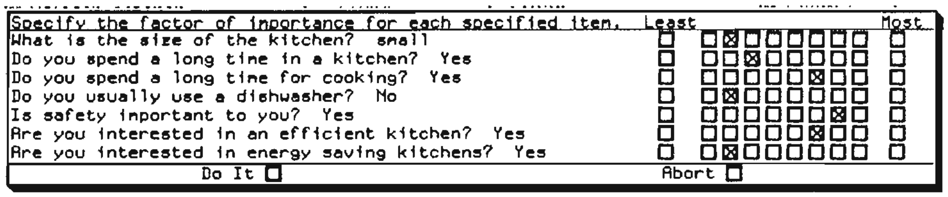
Specification sheet.

What is the size of the kitchen?	[✓ small]	large	Do-Not-Care
Do you spend a long time in a kitchen?	[✓ Yes]	No	Do-Not-Care
Do you spend a long time for cooking?	[✓ Yes]	No	Do-Not-Care
Do you usually use a dishwasher?	Yes	[✓ No]	Do-Not-Care
Is safety important to you?	[✓ Yes]	No	Do-Not-Care
Is light important to you?	Yes	No	[✓ Do-Not-Care]
Are you interested in easy plumbing?	Yes	No	[✓ Do-Not-Care]
Do you have to consider building codes?	Yes	Not so much	[✓ Do-Not-Care]
Are you interested in an efficient kitchen?	[✓ Yes]	No	Do-Not-Care
Are you interested in energy saving kitchens?	[✓ Yes]	No	Do-Not-Care

Done Abort

Figure 7: Specification Sheet

The *Specify* command in CATALOGEXPLORER provides a specification sheet in the form of a questionnaire. The questions are derived by analyzing questionnaires being used by professional kitchen designers.

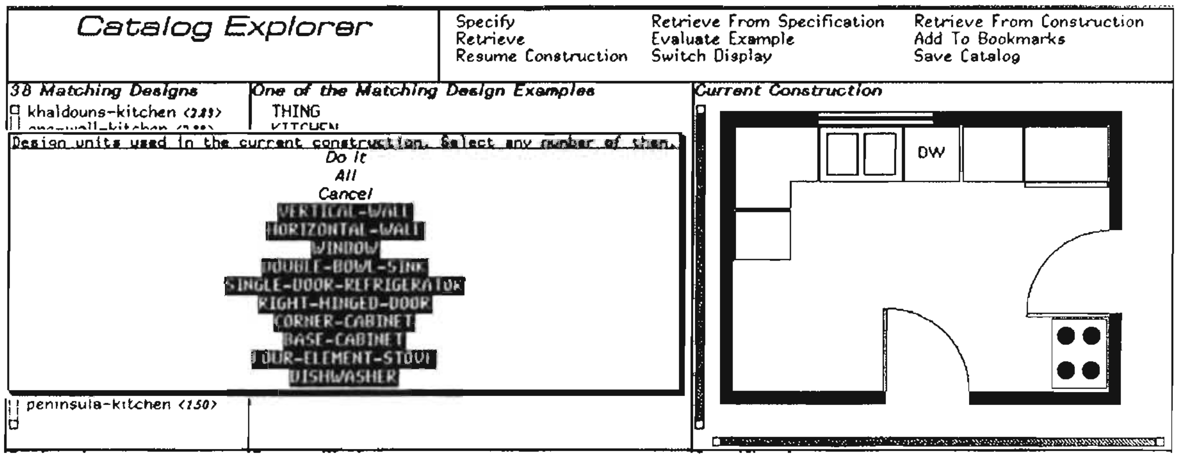


Specify the factor of importance for each specified item.	Least									Most
What is the size of the kitchen? small	☐	☐	☒	☐	☐	☐	☐	☐	☐	☐
Do you spend a long time in a kitchen? Yes	☐	☐	☐	☒	☐	☐	☐	☐	☐	☐
Do you spend a long time for cooking? Yes	☐	☐	☐	☐	☐	☐	☒	☐	☐	☐
Do you usually use a dishwasher? No	☐	☐	☒	☐	☐	☐	☐	☐	☐	☐
Is safety important to you? Yes	☐	☐	☐	☐	☐	☐	☐	☒	☐	☐
Are you interested in an efficient kitchen? Yes	☐	☐	☐	☐	☐	☐	☒	☐	☐	☐
Are you interested in energy saving kitchens? Yes	☐	☐	☒	☐	☐	☐	☐	☐	☐	☐

Do It ☐ Abort ☐

Figure 8: Weighting Sheet for the Specification

After specification, users are asked to weight the importance of each specified item.



Catalog Explorer

Specify
Retrieve
Resume Construction
Retrieve From Specification
Evaluate Example
Switch Display
Retrieve From Construction
Add To Bookmarks
Save Catalog

38 Matching Designs
khaldouns-kitchen ‹2**›
peninsula-kitchen ‹150›

One of the Matching Design Examples
THING
KITCHEN

Current Construction

Design units used in the current construction. Select any number of them.
Do It
All
Cancel
VERTICAL-WALL
HORIZONTAL-WALL
WINDOW
DOUBLE-BOWL-SINK
SINGLE-DOOR-REFRIGERATOR
RIGHT-HINGED-DOOR
CORNER-CABINET
BASE-CABINET
FOUR-ELEMENT-STOVE
DISHWASHER

Figure 9: *Retrieve From Construction*

The *Retrieve From Construction* command with a *parsing topic* "Design Unit Types" analyzes the current construction and provides a list of all the design unit types being used in the construction. Users can then select which design unit types they consider to be most important for locating prestored designs in the catalog.

---

of effort. If no relevant objects are found, they can modify the specification by either selecting other answers in the specification sheet, or changing the weights in the weighting sheet, or both. After this is done, the *Retrieval from Specification* command will reorder the examples. Users can also use the *Retrieval from Construction* command and choose other criteria for defining the similarity, which will retrieve another set of examples.

Finally, they can decide which example they want to use by bringing it into the *One of the Matching Design Examples* window, and go back to JANUS-CONSTRUCTION with the *Resume Construction* command. JANUS-CONSTRUCTION automatically will show their selected example in the *Catalog* window of JANUS-CONSTRUCTION (Figure 3). Users can refer to this example for getting new ideas on how to proceed with their construction, or they may replace the current construction with the example found.

RETRIEVAL FROM SPECIFICATION

Issues related to specification

In order to use a partial specification for identifying a relevant design object, one must consider the following issues: types of specifications, weighting importance, and multiple contradictory features.

Types of specifications. We have observed that there exist two types of specifications for a design: *surface features* and *hidden features.* For example, the specification ''*a kitchen that has a dishwasher*'' is a surface feature that explicitly describes the design, whereas ''*a kitchen that has less than 100 square feet,*'' or ''*a kitchen good for a large family*'' are hidden features of the design that are not explicitly expressed in the final design artifact (Kolodner, 1990). Surface features are determined by the structure of a design, whereas hidden features are related to functions of the design rather than to the structure (Gero, 1990). Hidden features can be computed or inferred only by using domain knowledge. There are two types of specifications in hidden features, per se. Features such as ''*a kitchen that has less than 100 square feet*'' are objective or judgmental, whereas features such as ''*a kitchen good for a large family*'' are subjective. A set of formal rules can be defined for deriving objective hidden features. In contrast, subjective hidden features can be inferred only relative to one's viewpoint. An inference of whether a kitchen design is good for a large family is subject to dispute and may vary across time and society.

In practice, initial customer questionnaires that professional kitchen designers give to their customers often ask questions relating to subjective hidden feature specifications. The experts map these specifications to concrete structural features by using their domain knowledge and experience.

Mechanisms for retrieval of design objects from specifications should, therefore, be different according to their types. Retrieving design examples from the catalog by surface feature specification can be done with a conventional query mechanism. In contrast, for retrieving design examples by hidden feature specifications, the system must have the domain knowledge to infer those features.

Weighting importance. Sometimes specified items contradict each other. If those contradictions are among hidden features, users may not notice the occurrence of the contradictions. Consequently, the system cannot retrieve design examples from the catalog that satisfy their specification because there do not exist such examples. For example, consider the two specifications ''*a safe kitchen*'' and ''*a kitchen that provides easy access to the dining area.*'' Although they seem not to contradict each other, they do so in terms of hidden features. As seen in Figure 4, a stove should be away from a door for the first specification, whereas a stove should be close to a door for the second one.

To resolve the contradiction, users must prioritize the specifications and make tradeoffs. They have to differentiate importance of the specifications by assigning a weight to each specification item. If users specify that *"a safe kitchen"* is more important to them, kitchen designs in which the stove is away from a door are more appropriate to the user's specification than others.

Multiple contradictory features. One design object may have multiple contradictory features; that is, hidden features that semantically contradict each other. For example, there can be a kitchen design in which some relationships of appliances in the example are *good for a large family*, whereas other relationships in the design are *bad for a large family*. In practice, some part of a design may serve contradictory purposes to other parts of the same design.

Mechanisms

Specification-linking rules. CATALOGEXPLORER dynamically infers subjective hidden features of design examples in the catalog by using domain knowledge in the form of *specification-linking rules*. The *specification-linking rules* link each subjective hidden feature specification item to a set of physical condition rules. For example, in the middle of the *Specification* window in Figure 6 two rules are shown (*stove away from a door* and *a stove away from a window*), which are conditions for a kitchen to have a hidden feature *"a safe kitchen."*

Previous versions of CATALOGEXPLORER required design examples to have explicitly specified values for *good-for* and *bad-for* slots to represent subjective hidden features. This approach relied on the questionable assumption that one could determine a priori that these features will become relevant later. Such features may become obsolete under new circumstances (e.g., an inefficient kitchen design may become efficient by introducing new appliances such as a microwave). Designers cannot articulate all the subjective features of a design, and even if they could do so, such features may be difficult to understand.

The most important aspect of the *specification-linking rules* is that they can be dynamically derived from the content of JANUS-ARGUMENTATION. Suppose the system has the following internal representation for the *"(Fire Hazard)"* argument shown in Figure 4.

$$\neg \text{ (Away-from-p STOVE DOOR)} \rightarrow \text{'FIRE-HAZARDOUS}^1 \qquad (1)$$

And the system has the domain knowledge:

$$\text{'SAFETY} \rightarrow \neg \text{ 'FIRE-HAZARDOUS}^2 \qquad (2)$$

When users specify that they are concerned about safety, the system infers that design examples with a stove away from a door are appropriate to their need by the following inference. First, (1) is equivalent to the following:

[1]Symbols such as "FIRE-HAZARDOUS" and "SAFETY" represent concepts as constant values, whereas "STOVE" and "DOOR" represent classes of design units. "Away-from-p" is a predefined predicate computing a distance between two design units and returns true if it exceeds a certain amount.

[2]This should read as "For a kitchen to be safe, it needs to be *not* fire-hazardous."

$\neg$ 'FIRE-HAZARDOUS $\rightarrow$ (Away-from-p STOVE DOOR) (3)

Therefore, by using (2) and (3),

(2) $\wedge$ (3) $\rightarrow$ ('SAFETY $\rightarrow$ (Away-from-p STOVE DOOR)) (4)

Appropriateness to a set of specifications. To deal with some of the issues mentioned earlier, CATALOGEXPLORER provides a mechanism for assigning a weight to each specification item and uses the concept of *appropriateness* of a design example to a set of specification items. The appropriateness of a design in terms of a set of specification items is defined as in Figure 10.

$S_1, S_2, \ldots, S_n$ is a set of specification items with weights $w_1, w_2, \ldots, w_n$, respectively. For each specification item S_i, let $R_{ij}(j=1 \ldots m_i)$ be a set of physical necessary conditions specified by a specification-linking rule. Let E be an example design, and define E(R) as follows:

$$E(R)=\begin{cases} 1 & \text{if the condition R is satisfied in E} \\ 0 & \text{otherwise} \end{cases}$$

Then, the *appropriateness* of design E in terms of a set of specifications $S=\{(S_1,w_1),(S_2,w_2), \ldots, (S_n,w_n)\}$ is defined as follows:

$$\sum_{i=1}^{n} \{(\sum_{j=1}^{m_i} E(R_{ij})/m_i) \times w_i\}$$

Figure 10: Definition of the *Appropriateness* of a Design

As a simple example, suppose a user specified one item: *"Is safety important to you? YES"* with a weight 0.8. The physical necessary conditions of this item are *"a stove is away from a door"* and *"a stove is away from a window,"* as seen in the *Specification* window in Figure 6. Therefore, a kitchen that has a stove away from a door but close to a window gets the appropriateness value of 0.4=(1+0)/2×0.8.

RETRIEVAL FROM CONSTRUCTION

For retrieving design examples related to a partial construction, one must deal with the issues of matching design examples in terms of surface features of a design, namely, at a structural level. The issues discussed in the previous section, such as partial matching and factor of importance, also hold here.

Domain-specific parsers analyze the design under construction. They represent the user's criteria for the articulation of the task at hand from a partial construction. In other words, they determine how to define similarities between the partial construction and a design example in the catalog for retrieval of design examples from the catalog.

CATALOGEXPLORER supports the following two parsers. Users have a mechanism to choose which parser they want to use.

- *Design unit types*: Search for examples that have the same design unit types as the current construction. The system first analyzes the current construction, finds which design unit types are used, and provides the user with a menu to select some of them (see Figure 9).
- *Configuration of design units*: Search for examples that have the same configuration of design units. For example, if the current construction has a dishwasher next to a sink, the examples matching this configuration element will be retrieved.

RELATED WORK

Using catalogs in design raises many problems in *case-based reasoning*. Retrieval techniques used in case-based reasoning systems, however, are often applicable only for domains in which problems can be clearly articulated, such as word pronunciation (Stanfill and Waltz, 1988). Such systems do not support dealing with fluctuation of the problem specification and are inadequate for ill-defined problems.

In JULIA (Kolodner, 1988), problem and solution structures must be articulated in frame representations before starting a retrieval process. *Value Frames* used in JULIA provide the rationale behind a design decision, which can be used for the retrieval of cases. CATALOGEXPLORER needs to integrate mechanisms to support recording of the design rationale for this purpose (Fischer et al., 1991).

Most of case-based reasoning systems require representations of cases to be predetermined, and therefore are not feasible. The approach presented in this paper addresses an indexing problem (Kolodner, 1990) by using more than surface representation of a case and enables the match at more abstract levels of representations. Use of the specification-linking rules can be regarded as a type of analogical matching such as *systematicity-based match* in CYCLOPS (Navinchandra, 1988). In CYCLOPS, however, the explanations associated with cases must be predetermined and cannot be dynamically computed.

The INTERFACE system (Riesbeck, 1988) provides interesting mechanisms for addressing some of the issues relating to matching rules. One of them is the use of *abstraction hierarchies* for dealing with the issue of partial matching, which could be used in CATALOGEXPLORER to support retrieval from construction. Another mechanism is to differentiate the importance of design features. This is similar to the *weighting sheet* in CATALOGEXPLORER, but it requires the features to be linearly ordered. Assigned importance values in our system enable users to deal with more complex contradictory features. Being built for the purpose of constructing a case-based library, the INTERFACE system supported these mechanisms only while storing cases in the library. In our work, the retrieval processes are driven by the user's task at hand, requiring that the weights are determined at the retrieval time rather than at the time when cases are stored. The INTERFACE system supports the creation of such matching rules only in an ad hoc manner. The integrated architecture of CATALOGEXPLORER enables the specification-linking rules to be derived from the argumentation component associating the rules with a clearly stated rationale.

The system allows users to store design examples in the catalog without checking for duplications and redundancies. Other systems store only prototypes (Gero, 1990), or prototypes and a small number of examples that are a variation of them (Riesbeck, 1988).

These approaches allow users to access *good* examples easily and prevent the chaotic growth of the size of the catalog. However, by not including failure cases, these catalogs prevent users from learning what went wrong in the past.

Many case-based reasoning systems support comprehension and adaptation of cases (Kolodner, 1990). CATALOGEXPLORER supports the comprehension of examples by allowing users to evaluate them with CONSTRUCTION ANALYZER. Adaptation is done by the users by bringing an example into the *Work Area* in JANUS-CONSTRUCTION. No efforts have been made toward automating adaptation in our approach.

DISCUSSION

Achievements

By integrating knowledge-based construction, hypermedia argumentation, catalogs of prestored design objects, and specification components, several crucial design activities can be supported, such as relevance to the task at hand, the situation talking back, reflection-in-action (Schoen, 1983), and integration of problem analysis and synthesis.

In CATALOGEXPLORER, users gradually narrow a catalog space. The system can dynamically infer subjective hidden features and provide users with an explanation for the inference mechanism. The system retrieves examples similar to the current construction, providing users further directions in proceeding the design or warning them of potential failures. Using the retrieved information they can incrementally evolve a specification and a construction in JANUS. The retrieval mechanisms of the system allow users to access information relevant to the task at hand in a more effective and accurate way without requiring the users to form queries. Control and responsibility of retrieval of information is shared between the user and the system (Fischer, 1990).

Limitations

A major limitation of the current system is the relatively small size of the catalog (less than a hundred examples). Many problems of managing large spaces effectively have not been dealt with. A lack of mechanisms for associating formal representations to arguments forces us to manually derive the *specification-linking rules.* The definition of appropriateness is limited and needs a more sophisticated mechanism such as connectionist networks (Henninger, 1990). The parsers for analyzing partial constructions should be extended to deal with more abstract levels, such as an emerging shape (e.g., L-shape or U-shape) that currently requires to be specified by the user. A combinatorial use of the parsers should be explored.

Future work

Future extensions of integrated design environments based on the multifaceted architecture include:

- *Level of Assembly.* The use of JANUS by kitchen designers has illustrated that the designers work not only with design units, but with higher level abstractions such as cooking centers and clean-up centers. These centers should be integrated into the palette, eliminating clear distinction between the elements in the palette and the catalog. The catalog should contain not only completed designs, but also important partial designs. These extensions will require further consideration on issues such as how to focus on a solution (Kolodner, 1990).

- *Support for Other Transition Links.* A partial specification can be used to determine the set of relevant arguments in the argumentation component, enabling us to dynamically rearrange argumentation space. A link between construction and specification can reduce the set of relevant units displayed in the palette.
- *Extensions of the Architecture.* Our design environment for user interface design (Lemke, 1989; Lemke and Fischer, 1990) has been improved greatly in its effectiveness by having a *checklist* component to help users to structure and organize their design activities. The integration of the checklist into the multifaceted architecture has to be explored further.
- *End-User Modifiability.* In developing design environments, domain knowledge should be built into a *seed.* As users use the environment continually, this seed should be extended. Sophisticated mechanisms for end-user modifiability (Fischer and Girgensohn, 1990) are crucial for this evolution of seeded environments.

CONCLUSION

Design activities incorporate many cognitive issues such as recognizing and framing a problem, understanding given information, and adapting the information to the situation. Integration of problem setting and problem solving is crucial in dealing with ill-defined problems. In this paper we have described mechanisms relating partial specifications and partial constructions to a catalog of prestored designs, thereby retrieving design objects stored in a catalog relevant to the task at hand without asking users to form queries. The system demonstrates the synergy of integrated design environments empowering human designers. The multifaceted architecture developed in the context of these research efforts is a promising architecture for building a great variety of integrated design environments in different domains.

ACKNOWLEDGMENTS

The authors would like to thank the members of the Human-Computer Communication group at the University of Colorado, who contributed to the conceptual framework and the systems discussed in this paper. The research was supported by Software Research Associates, Inc. (Tokyo, Japan), by the National Science Foundation under grants No. IRI-8722792 and IRI-9015441, and by the Army Research Institute under grant No. MDA903-86-C0143.

REFERENCES

Cross, N. (1984). *Developments in Design Methodology,* John Wiley & Sons, New York.

Fischer, G. (1989). *Creativity Enhancing Design Environments,* Proceedings of the International Conference 'Modelling Creativity and Knowledge-Based Creative Design' (Heron Island, Australia), pp. 127-132.

Fischer, G. (1990). *Communications Requirements for Cooperative Problem Solving Systems,* The International Journal of Information Systems (Special Issue on Knowledge Engineering), Vol. 15, No. 1, pp. 21-36.

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

Fischer, G., and Lemke, A.C. (1988). *Construction Kits and Design Environments: Steps Toward Human Problem-Domain Communication*, Human-Computer Interaction, Vol. 3, No. 3, pp. 179-222.

Fischer, G., and Nieper-Lemke, H. (1989). *HELGON: Extending the Retrieval by Reformulation Paradigm*, Human Factors in Computing Systems, CHI'89 Conference Proceedings (Austin, TX), ACM, New York, pp. 357-362.

Fischer, G., Lemke, A.C., Mastaglio, T., and Morch, A. (1990). *Using Critics to Empower Users*, Human Factors in Computing Systems, CHI'90 Conference Proceedings (Seattle, WA), ACM, New York, pp. 337-347.

Fischer, G., Lemke, A.C., McCall, R., and Morch, A. (1991). *Making Argumentation Serve Design*, Technical Report, Department of Computer Science, University of Colorado, Boulder, CO.

Fischer, G., Henninger, S., and Redmiles, D. (1991). *Intertwining Query Construction and Relevance Evaluation*, Human Factors in Computing Systems, CHI'91 Conference Proceedings (New Orleans, LA), ACM, New York, (in press).

Fischer, G., McCall, R., and Morch, A. (1989). *JANUS: Integrating Hypertext with a Knowledge-Based Design Environment*, Proceedings of Hypertext'89 (Pittsburgh, PA), ACM, New York, pp. 105-117.

Gero, J.S. (1990). *Design Prototypes: A Knowledge Representation Schema for Design*, AI Magazine, Vol. 11, No. 4, pp. 26-36.

Halasz, F.G. (1988). *Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems*, Communications of the ACM, Vol. 31, No. 7, July, pp. 836-852.

Henninger, S. (1990). *Defining the Roles of Humans and Computers in Cooperative Problem Solving Systems for Information Retrieval*, Proceedings of the AAAI Spring Symposium Workshop on Knowledge-Based Human-Computer Communication, pp. 46-51.

Hutchins, E.L., Hollan, J.D., and Norman, D.A. (1986). *Direct Manipulation Interfaces*, in D.A. Norman and S.W. Draper (eds.), *User Centered System Design, New Perspectives on Human-Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, NJ, pp. 87-124, ch. 5.

Kolodner, J.L. (1988). *Extending Problem Solving Capabilities Through Case-Based Inference*, in J. Kolodner (ed.), Proceedings: Case-Based Reasoning Workshop, Morgan Kaufmann Publishers, Clearwater Beach, FL, pp. 21-30.

Kolodner, J.L. (1990). *What is Case-Based Reasoning?*, Tutorial Text on Case-Based Reasoning, AAAI'90, Boston, MA, pp. 1-32.

Lemke, A.C. (1989). *Design Environments for High-Functionality Computer Systems*, Unpublished Ph.D. Dissertation, Department of Computer Science, University of Colorado.

Lemke, A.C., and Fischer, G. (1990). *A Cooperative Problem Solving System for User Interface Design*, Proceedings of AAAI-90, Eighth National Conference on Artificial Intelligence, AAAI Press/The MIT Press, Cambridge, MA, pp. 479-484.

McCall, R. (1986). *Issue-Serve Systems: A Descriptive Theory of Design*, Design Methods and Theories, Vol. 20, No. 8, pp. 443-458.

McCall, R., Fischer, G., and Morch, A. (1990). *Supporting Reflection-in-Action in the Janus Design Environment*, in M. McCullough et al. (eds.), *The Electronic Design Studio*, The MIT Press, Cambridge, MA, pp. 247-259.

Navinchandra, D. (1988). *Case-Based Reasoning in CYCLOPS*, in J. Kolodner (ed.), Proceedings: Case-Based Reasoning Workshop, Morgan Kaufmann Publishers, Clearwater Beach, FL, pp. 286-301.

Nielsen, J., and Richards, J.T. (1989). *The Experience of Learning and Using Smalltalk*, IEEE Software, May, pp. 73-77.

Reeves, B. (1990). *Finding and Choosing the Right Object in a Large Hardware Store -- An Empirical Study of Cooperative Problem Solving among Humans*, Technical Report, Department of Computer Science, University of Colorado, Boulder, CO.

Riesbeck, C.K. (1988). *An Interface for Case-Based Knowledge Acquisition*, in J. Kolodner (ed.), Proceedings: Case-Based Reasoning Workshop, Morgan Kaufmann Publishers, Clearwater Beach, FL, pp. 312-326.

Riesbeck, C.K., and Schank, R.C. (1989). *Inside Case-Based Reasoning*, Lawrence Erlbaum Associates, Hillsdale, NJ.

Rissland, E.L., and Skalak, D.B. (1989). *Combining Case-Based and Rule-Based Reasoning: A Heuristic Approach*, Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (Detroit, MI), Morgan Kaufmann Publishers, Palo Alto, CA, pp. 524-530.

Rittel, H.W.J. (1984). *Second-generation Design Methods*, in N. Cross (ed.), *Developments in Design Methodology*, John Wiley & Sons, New York, pp. 317-327.

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

Sheil, B.A. (1983). *Power Tools for Programmers*, Datamation, February, pp. 131-144.

Simon, H.A. (1973). *The Structure of Ill-Structured Problems*, Artificial Intelligence, No. 4, pp. 181-200.

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

Stanfill, C., and Waltz, D.L. (1988). *The Memory-Based Reasoning Paradigm*, in J. Kolodner (ed.), Proceedings: Case-Based Reasoning Workshop, Morgan Kaufmann Publishers, Clearwater Beach, FL, pp. 414-424.

Stefik, M.J. (1986). *The Next Knowledge Medium*, AI Magazine, Vol. 7, No. 1, Spring, pp. 34-46.

Williams, M.D. (1984). *What Makes RABBIT Run?*, International Journal of Man-Machine Studies, Vol. 21, pp. 333-352.

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