AMPLIFYING DESIGNERS' CREATIVITY WITH DOMAIN-ORIENTED DESIGN ENVIRONMENTS

GERHARD FISCHER
University of Colorado
and
KUMIYO NAKAKOJI
University of Colorado and Software Research Associates, Inc., Tokyo

1. Introduction

Design (Simon, 1981) is one of the most promising areas in which to study creativity, because of the following features of design problems:

- designers tackling the same problem are likely to come up with different solutions (Jacob, 1977);
- good designers break rules all the time;
- design deals with ill-defined (Simon, 1981) and wicked problems (Rittel, 1984) (i.e. problems that are intrinsically open-ended, situation specific and controversial); and
- in design there are no optimal solutions, but only trade-offs.

The research discussed in this paper is based on the assumption that design problems are best solved by fostering co-operative problem-solving between humans and integrated, domain-oriented, knowledge-based design environments (Fischer, 1990). Combining knowledge-based systems and innovative human-computer communication techniques empowers designers to produce 'better' products by amplifying their creative skills (Fischer, 1989).

Our approach is *not* to build another expert system. Expert systems require an adequate understanding of a problem to begin with. The relevant factors and background knowledge need to be identified. In design domains this information cannot be fully articulated. What has been made explicit always sets a limit, and there is always the possibility that breakdowns will require us to go beyond this limit (Winograd and Flores, 1986).

In this paper we use the domain of the architectural design of kitchen floor plans as an 'object-to-think-with', for the purposes of illustration (Fischer, McCall et al., 1989). The familiarity and simplicity of this domain helps us to concentrate on the essential issues of our approach without being distracted by understanding the semantics of the particular domain. We first discuss general issues of design environments, and emphasize the importance of domain orientation and the integration of tools that support different aspects of design. We then describe the

T. Dartnall (ed.), Artificial Intelligence and Creativity, 343–364.

multifaceted architecture that underlies these environments. This gives us a conceptual framework for our research. The environments support four design themes that are important for supporting creative design:

- co-evolution of problem specification and solution construction;
- reflection in action;
- evolution of design environments;
- making relevant information available.

CatalogueExplorer, an innovative systems component, illustrates how integrated environments can amplify human creativity in terms of the fourth theme. It integrates specifications, constructions, and a catalogue of pre-stored design objects. The synergy of this integration enables the system to retrieve design objects that are relevant to the task at hand, as identified by a partial specification and a partial construction, thereby notifying designers of the existence of potentially relevant information. By presenting information to designers that they may never have thought of, the mechanism amplifies their creativity by "bringing existing design concepts into unseen and even unthought, yet valuable ways of usage" (McLaughlin and Gero, 1989). It is up to the designers whether or not to relate this new information to the task at hand. This emphasizes the basic assumption that creativity is not just a mental capacity (Boden, 1990), but is greatly enhanced by interacting—in the right way—with knowledge in the world (Norman, 1993).

2. Problems

This section outlines some of the problems our research addresses in creating environments that amplify human creativity.

2.1. Integrating problem setting and problem solving

The integration of problem setting and problem solving is indispensable in dealing with ill-defined design problems (Schön, 1983). Complex designs are implemented over a long period of time and are modified throughout the design process (Simon, 1981). Simon states that they have much in common with painting in oil, where current goals lead to new applications of paint, and where the gradually changing pattern suggests new goals. We cannot gather information unless we have understood the problem, and we cannot understand the problem without having information about it. Professional practitioners have at least as much to do with defining the problem as they do with solving it (Rittel, 1984).

An empirical study by our research group, which analysed *human-human cooperative problem solving* between customers and sales agents in a large hardware store (Fischer and Reeves, 1992), provided ample evidence that in many cases people are initially unable to specify complete requirements for ill-defined problems. They start with a partial specification and refine it incrementally, in terms of the feedback they get from their environment.

In design, this feedback is provided by "the back talk of the situation" (Schön, 1983). While engaging in a *conversation with the design material*, designers become aware of an occurrence of a breakdown. This awareness is triggered by an evaluation of the current design stage in terms of the task at hand. The evaluation is carried out either by the designers themselves, or by outside agents, such as design teachers or specialists in computer-supported design environments (Fischer, Lemke, Mastaglio et al., 1991). Reflection upon the situation results in determining the next move in problem setting and/or problem solving.

2.2. DOMAIN ORIENTATION

To turn computers into a design medium for domain-oriented professionals, we have to reduce the gap between a computational design substrate and an application domain (Hutchins, Hollan, and Norman, 86). Designers should perceive design as communication with an application domain, rather than as manipulating symbols on a computer display. 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 users to design artifacts from applications-oriented building blocks of various levels of abstraction.

2.3. ARTICULATING THE TASK AT HAND

To support the integration of problem setting and problem solving in design environments, it is crucial to identify information that is relevant to the task at hand (Fischer and Nakakoji, 1991). Every step made by a designer towards a solution determines a new space of related information, which cannot be determined *a priori*, by its very nature. Integrated design environments are based on high-functionality systems (Lemke, 1989) that contain a large number of design objects. Such systems increase the likelihood of an object existing that is close to what is needed, but, without adequate systems support, it is difficult to locate and understand such objects (Nielsen and Richards, 1989; Fischer, Henninger et al., 1992). Suppose that a designer wants to design a floor plan for a safe kitchen that is suitable for a left-handed person with a small child. Given hundreds of fancy pictures of kitchen floor plans in a catalogue, it is difficult for the designer to access the information that is relevant to the present task (see Figure 1). Conventional information access techniques, such as queries and browsing, often support a decontextualized information need. Although the context can, in principle, be explicitly stated in queries or dealt with by browsing techniques, this greatly complicates the information retrieval process in design environments. Query-based access mechanisms require users to articulate exactly what they are looking for by formulating highly specific queries. The *navigational access* provided by browsing mechanisms places most of the burden of traversing the information space on users, who tend to get lost in large, complex spaces (Halasz, 1988).

Information needs in design environments arise against a background of concerns about the larger context of the problem that needs to be solved (Fischer, Henninger et al., 1992). The *task at hand* is articulated both by the partially constructed artifact, and by an evolving requirement specification that represents the high-level design concerns of the artifact.

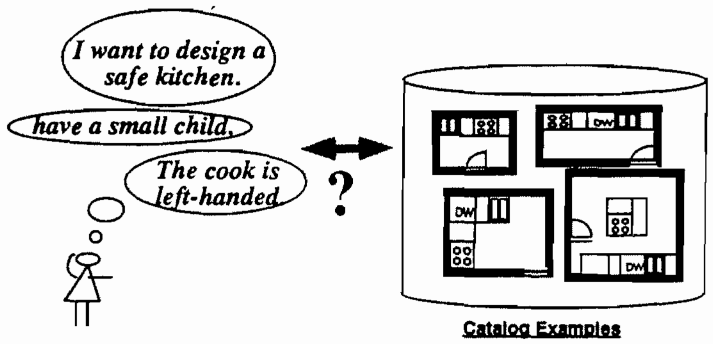


Fig. 1. Location of information relevant to the task at hand.
Conventional query-based and navigational searches do not help designers to access information in the catalogue that is relevant to the task at hand.

3. A multifaceted architecture for integrated design environments

Design is a conversation with the materials of a design situation. This principle has been operationalized by creating domain-oriented design environments that support human problem-domain communication (Fischer and Lemke, 1988). The 'materials' of the design situation are not low-level computer abstractions but objects with which the designer is familiar. The domain-oriented nature of the environments acknowledges the fact that knowledge does not exist by itself in the form of context-free information, but is part of the practice of specific professional communities (Ehn, 1988).

Over the last five years, we have developed and evaluated several prototype domain-oriented design environments (Fischer, McCall et al., 1989; Lemke and Fischer, 1990; Fischer, Grudin et al., 1992). Figure 2 shows the domain-independent components of the multifaceted architecture that are instantiated with domain-specific information when a domain-oriented design environment is created. We will describe these components in the context of the Janus system, which supports the design of kitchen floors (Fischer, McCall et al., 1989). The system is implemented in Common Lisp, and runs on Symbolic Lisp machines.

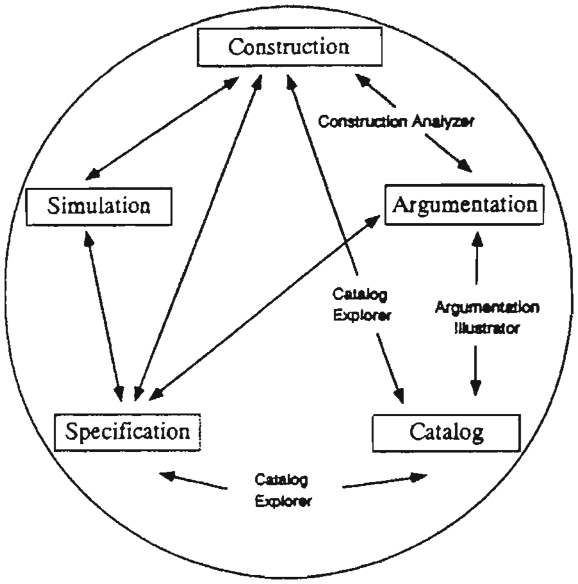
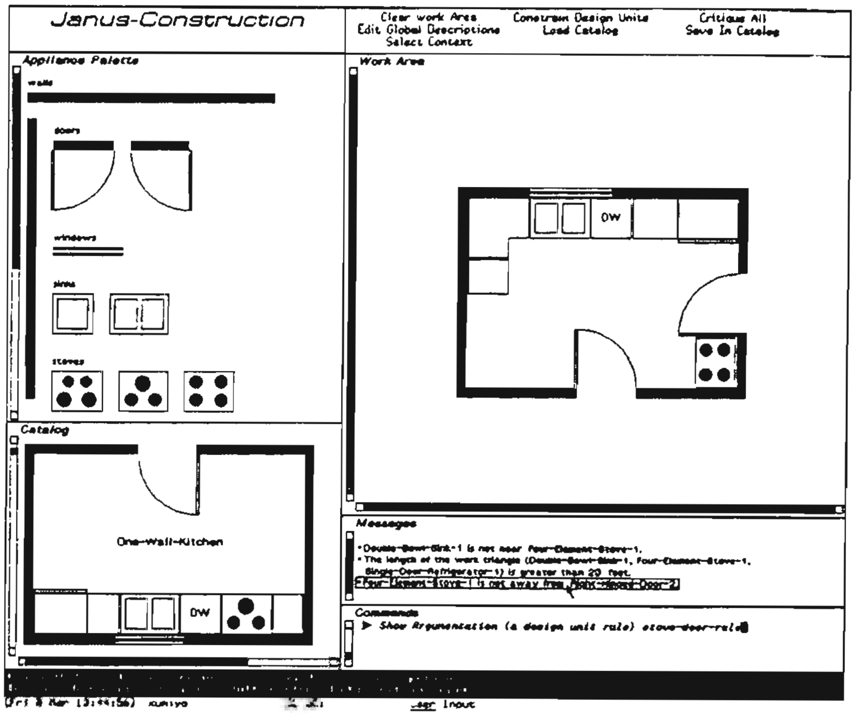


Fig. 2. The components of the multifaceted architecture for an integrated design environment. The links between the components are crucial for the synergy of integration.

3.1. Components of the multifaceted architecture

Integrated design environments that are based on the multifaceted architecture are composed of the following interface components (Figure 2):

- A *construction kit* (see Figure 3) is the principal medium for the implementation of the design. It provides a palette of domain abstractions, and supports the construction of artifacts by direct manipulation and other interaction styles. A specific construction (as seen in the Work Area pane) represents a concrete implementation of a design, and reflects the user's current problem situation.
- A *specification component* (see Figure 4) allows designers to describe some of the required characteristics of the design at a high level of abstraction. It assigns weights of importance to each specified item. The specifications are expected to be modified and augmented during the design process, rather than being fully articulated at the beginning. They are used to prioritize the information spaces in the system with respect to the emerging task at hand.
- An *issue-based argumentative hypermedia system* (see Figure 5) captures the design rationale. Information fragments in the hypermedia issue base are based on an issue-based information system (McCall, 1986) and are linked according to whatever information resolves an issue that is relevant to a partial construction (Fischer, Lemke, McCall et al., 1991). The issues, answers and



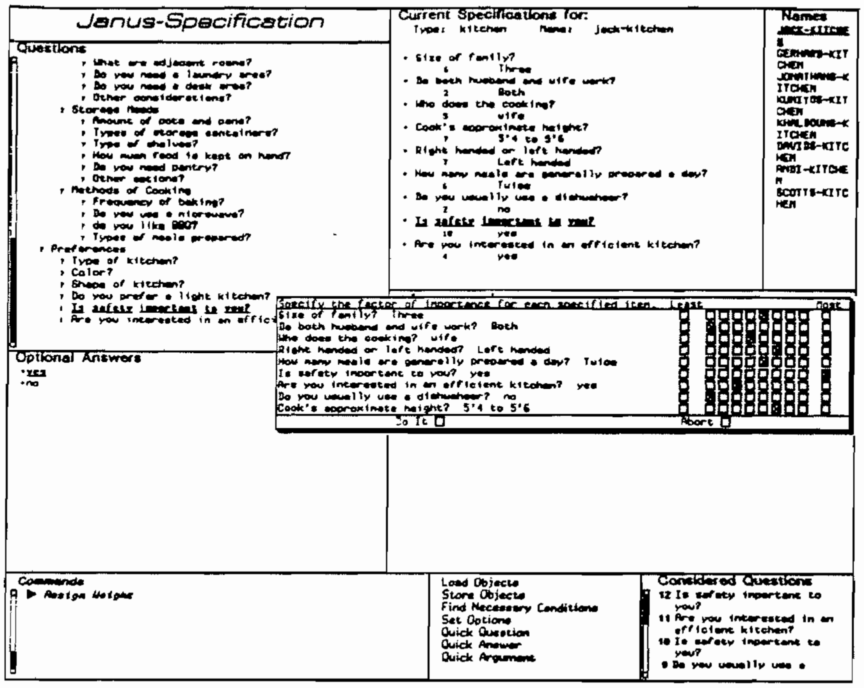
• Double-Bowl-Sink-1 is not near Four-Element-Stove-1.
• The length of the work triangle (Double-Bowl-Sink-1, Four-Element-Stove-1, Single-Door-Refrigerator-1) is greater than 20 feet.
• Four-Element-Stove-1 is not away from Right-Hinged-Door-2.

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

Fig. 3. Screen image of Janus Construction.
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 automatically displays critiques after each design change that triggers a critic message. Clicking on a message activates Janus Argumentation and displays the argumentation underlying the message (see Figure 5).

arguments held in Janus Argumentation (see Figure 5) can be accessed via links from the domain knowledge in other components.

– A *catalogue* (see Figures 3 and 7) provides a collection of prestored design objects that illustrates the space of possible designs in the domain. Catalogue examples amplify a designer's creativity by providing new ideas and perspectives for the design.
– A *simulation component* allows the user to carry out 'what-if' games to simulate usage scenarios with the artifact being designed. Such simulation complements the argumentative component.



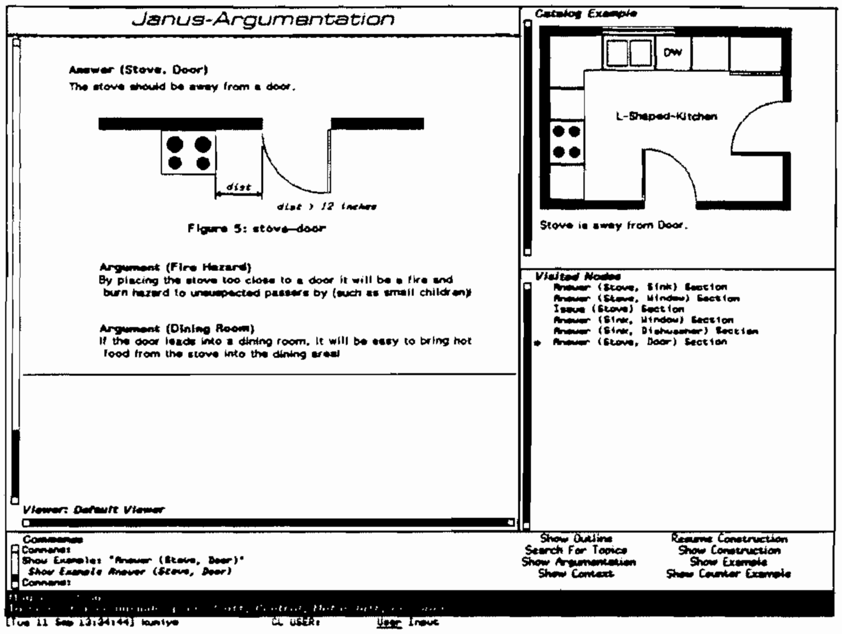
```
Janus-Specification

Questions
        What are adjacent rooms?
        Do you need a laundry area?
        Do you need a desk area?
        Other considerations?
      Storage Needs
        Amount of pots and pans?
        Types of storage containers?
        Type of shelves?
        How much food is kept on hand?
        Do you need pantry?
        Other sections?
      Methods of Cooking
        Frequency of baking?
        Do you use a microwave?
        do you like BBQ?
        Types of meals prepared?
    Preferences
      Type of kitchen?
      Color?
      Shape of kitchen?
      Do you prefer a light kitchen?
      Is safety important to you?
      Are you interested in an effic

Optional Answers
 *yes
 *no

Current Specifications for:
  Type: kitchen     Name:   jack-kitchen

 • Size of family?
      6         Three
 • Do both husband and wife work?
      2         Both
 • Who does the cooking?
      5         wife
 • Cook's approximate height?
      7         5'4 to 5'6
 • Right handed or left handed?
      7         Left handed
 • How many meals are generally prepared a day?
      6         Twice
 • Do you usually use a dishwasher?
      2         no
 • Is safety important to you?
      10        yes
 • Are you interested in an efficient kitchen?
      4         yes

Names
JACK-KITCHEN
GERHARDS-KITCHEN
JONATHANS-KITCHEN
KUMIYOS-KITCHEN
KHALBOUNS-KITCHEN
DAVIDS-KITCHEN
ANDI-KITCHEN
SCOTTS-KITCHEN

Specify the factor of importance for each specified item.  Least      Most
Size of family?  Three
Do both husband and wife work?  Both
Who does the cooking?  wife
Right handed or left handed?  Left handed
How many meals are generally prepared a day?  Twice
Is safety important to you?  yes
Are you interested in an efficient kitchen?  yes
Do you usually use a dishwasher?  no
Cook's approximate height?  5'4 to 5'6
              Do It                                 Abort

Commands
 ▶ Assign Weight

Load Objects
Store Objects
Find Necessary Conditions
Set Options
Quick Question
Quick Answer
Quick Argument

Considered Questions
12 Is safety important to
   you?
11 Are you interested in an
   efficient kitchen?
10 Is safety important to
   you?
 9 Do you usually use a
```

Fig. 4. Screen image of Janus Specification.
The *specify* command in CatalogueExplorer (see Figure 7) provides a specification sheet in the form of a questionnaire. After specification, users weigh the importance of each specified item.

3.2. Integration of the Multifaceted Architecture

The architecture derives its value from the integration of its components and the links between them. Used individually, the components cannot achieve their full potential, but used in combination they form a synergistic whole. Links between components are supported by a number of mechanisms (see Figure 2). These include:

– The Construction Analyser. This is a critiquing component (Fischer, Mastaglio et al., 1991) that detects and critiques partial solutions constructed by users. The firing of a critic signals a breakdown (Winograd and Flores, 1986), warning users 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 occurs (see Figures 3 and 5).
– The Argumentation Illustrator. This helps users to understand the information given in an argumentative hypermedium by providing an example (see Figure 5). Explanations given as argumentation are often highly abstract and very


Janus-Argumentation

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

Catalog Example

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:

[Tue 11 Sep 13:34:44] kumiyo CL USER: User Input

Fig. 5. Screen image of Janus Argumentation.
This shows an answer to the question of where to locate the kitchen stove with respect to a door. It shows the desirable relative positions of the two units. Below this is a list of arguments for and against the answer. The example in the upper right corner (which is activated by the *show example* command in the *Commands* pane) contextualizes an argumentative principle in relation to a specific design (carried out by the Argumentation Illustrator).

conceptual. Concrete design examples help users to understand the concepts.

– CatalogueExplorer. This is described in detail below. It helps designers to search the catalogue space and retrieve examples that are similar to the current construction situation. It orders the examples according to their relevance to the current specification.

4. Going beyond the macho approach of artificial intelligence with Domain-Oriented Design Environments

Our work addresses the problems mentioned above and asks how they can be facilitated by the computer-based tools of our multifaceted architecture. Our interest is in understanding how designers design (Fischer and Böcker, 1983), how they might organize their design activities so that they are more effective and less error-prone, how they learn new things as they go along, how they produce creative artifacts, and how all or some of these activities can be supported by computational media. Our thinking has been influenced by the work of a number of researchers who are trying to gain a deeper understanding of design as a creative activity (e.g., Simon, 1981; Schön, 1983; Rittel, 1984; Winograd and Flores, 1986; Suchman, 1987; Ehn, 1988; and Lave, 1988). Some of these methodologists, however, have not followed their own theories, since they have failed to intertwine theory building (reflection) with theory instantiation (action). We are engaged in building 'objects-to-think-with', in the forms of demonstration prototypes, to test our theories in practice, to experience breakdowns of a theory, and to refine it as a consequence. We try to demonstrate that computational mechanisms can be created that can take some of the concepts mentioned in Section 2, and bring them to life in a computational environment.

4.1. Design Activities Supported by Our Design Environments

Design environments based on the multifaceted architecture amplify the creative skills of designers by integrating a number of different aspects of design activity. We discuss four activities that our environment supports.

4.1.1. Co-evolution of problem specification and solution construction

Designers start with a vague design goal, and go back and forth between different components in our environment. A typical cycle of events in the environment includes the following:

- designers create a partial specification or a partial construction;
- they do not know how to continue, so
- they switch and consult other components in the system that provide them with information that is relevant to the partially articulated task at hand; then
- they refine their understanding by reflecting upon the situation.

As designers move between components, the problem space is narrowed and different facets of the artifact are refined.

4.1.2. Reflection in action

Design (as supported by the multifaceted architecture) iterates through cycles of specification, construction, evaluation, and reuse. At each stage, the partial design serves as a stimulus for suggesting what users should attend to next. The direction of a new move permits new information to be extracted from memory and reference sources, and leads to new steps toward the development of the design. The integration of various aspects of the design enables the situation to 'talk back' to users, following the characterization of design activity given by Schön (1983):

> 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].

We pay tribute to this concept by integrating construction and argumentation with the help of critics (Fischer, Lemke, Mastaglio et al., 1991; Fischer, Lemke, McCall et al., 1991).

4.1.3. Evolution of design environments

Design knowledge is tacit, and competent practitioners usually know more than they can say (Polanyi, 1966). Their tacit knowledge is triggered by new design situations and by breakdowns that occur as they engage in a design process. Design environments must be open-ended and modifiable by their users (Fischer and Girgensohn, 1990). In the Janus environment, a system component called Modifier (Fischer and Girgensohn, 1990) allows end-users (professional kitchen designers rather than software designers) to add domain concepts without dealing with the underlying programming language. A completed design artifact (consisting of a specification and a construction) may be stored in the catalogue for later reuse. If users do not agree with the argumentation presented to them, they can add their own counter-arguments in the argumentative hyper-media component. Through these processes, the environment gradually accumulates design knowledge through constant use (Henderson and Kyng, 1991; Fischer, Grudin et al., 1992).

4.1.4. Making information relevant to the task at hand

The integration provided by the multifaceted architecture enables the system to incrementally identify the task at hand. Suppose a user is designing a kitchen as shown in Figure 1. In this example, the partially articulated task is to design a floor plan for a kitchen that is suitable for a left-handed person with a small child. On the basis of this partial specification, the system provides users with relevant information without requiring them to form queries or navigate through large information spaces to locate relevant information. (This process is described in more detail in Section 5.) By implicitly creating queries, the system accesses relevant information that users may not have thought of. It is up to them whether to use this information, but it encourages them to view the current design from a new perspective.

4.2. AMPLIFYING THE DESIGNER'S CREATIVITY

To amplify creativity in design, our environments provide designers with information relevant to the task at hand. *Reminding* is considered to be crucial in supporting creativity (McLaughlin and Gero 1989; Boden, 1990). In our environments reminding is supported by:

– breakdowns, that signal the violation of a rule;

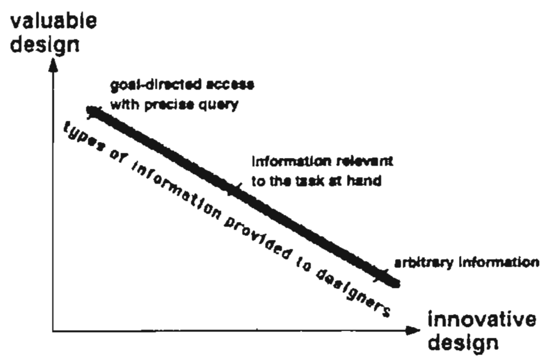

valuable design
goal-directed access with precise query
information relevant to the task at hand
arbitrary information
types of information provided to designers
innovative design

Fig. 6. Spectrum of information provided by design environments in relation to different dimensions of creativity.

– accessing domain concepts in the argumentation; and
– locating interesting examples in the catalogue.

Designers can analyse this information, and draw analogies and/or discover new ideas that will lead them to make new moves that they might otherwise have overlooked.

The challenge of building computational environments is primarily not just to provide more information, but to say the 'right' thing at the 'right' time (Fischer, Nakakoji et al., 1992). There is a spectrum of how accurately and closely related the presented information should be to the task at hand. At one end is information that is retrieved on the basis of precise questions formulated by the designer. At the other is information that is only relevant in a very general way (Owen, 1986). Creative design should be *innovative* and *valuable*. If designers are given arbitrary information, they may get innovative ideas, but these may not be valuable (though we should not overlook the value of serendipity (Roberts, 1989)). If they are given information that is based on precise queries, they may get valuable but less innovative ideas (see Figure 6).

By exploiting the information contained in a partial specification and a partial construction, our environments can provide information that can give rise to ideas that are both valuable and innovative. Our environments have three mechanisms for doing this:

– the Construction Analyser is the critiquing component (Fischer, Lemke, McCall et al. 91) that *signals breakdowns* (such as the violation of basic design principles);
– CatalogueExplorer *accesses* design examples relevant to the task at hand, and
– Case-Deliverer *delivers* catalogue design examples without an explicit request from the user (Fischer, Henninger et al., 1992).

By integrating construction and argumentation (Fischer, Lemke, McCall et al., 1991), we overcome the deficiencies of non-integrated systems (e.g., gIBIS (Conklin and Begeman, 1988)), where it is impossible to access information relevant to the task at hand.

5. CatalogueExplorer

In this section we describe CatalogueExplorer, which links the specification and construction components with the catalogue (see Figure 2). We provide a scenario that illustrates a typical use of the system, and then describe the mechanisms underlying the scenario.

5.1. SYSTEM DESCRIPTION

Design objects stored in a catalogue can be used for

- providing a solution to a new problem;
- warning of possible failures; and
- evaluating and justifying a decision (Kolodner, 1990; Rissland and Skalak, 1989).

The catalogue provides a source of potentially interesting ideas. Designers may be reminded of a new way of designing by drawing an analogy between a catalogue example and their current task. For large catalogues, however, identifying design examples that are relevant to the task at hand is a challenging and time-consuming task (see Figure 1).

By integrating specification, construction, and a catalogue, CatalogueExplorer helps users to retrieve information that is relevant to the task at hand. This helps them to refine their partial specification and partial construction, which eliminates the need to form queries or browse in the catalogue.

The design examples in the catalogue are stored as objects in the Kandor knowledge-base (Patel-Schneider, 1984). Each design example consists of a floor layout and a set of slot values. The examples are automatically classified according to their features specified as these slot values.

CatalogueExplorer (see Figure 7) 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 search iteratively for more appropriate design information by refining the specification, rather than being constrained by an initially specified query.

On the basis of the retrieval-by-reformulation paradigm, CatalogueExplorer retrieves design objects that are relevant to the task by:

- using the information contained in a partial specification to prioritize the designs stored in the catalogue (*retrieval from specification*)

- analysing the current construction, and retrieving *similar* examples from the catalogue using similarity metrics (*retrieval from construction*).

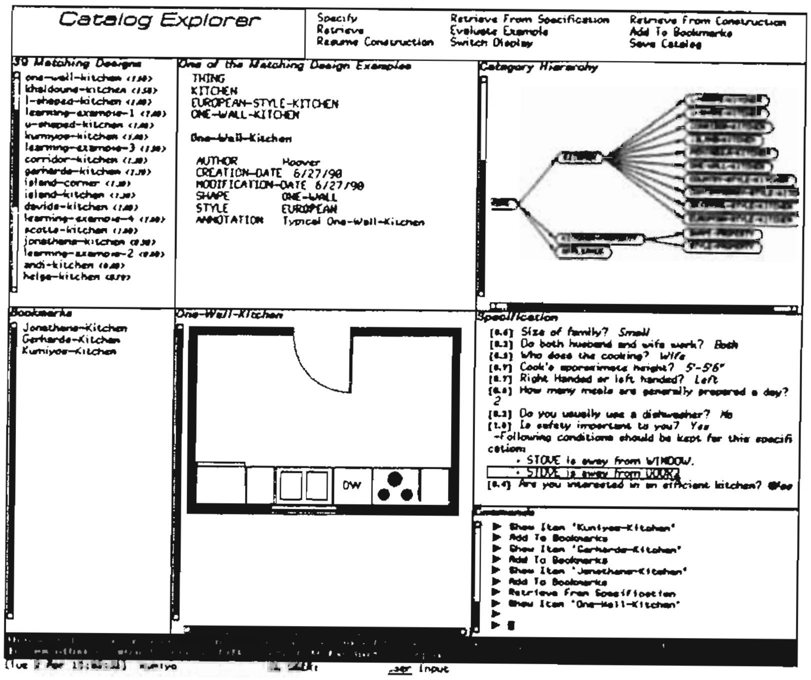
5.2. A SCENARIO USING CATALOGUEEXPLORER

CatalogueExplorer (see Figure 7) is invoked by the *catalogue* command from Janus Construction (Figure 3). The *specify* command invokes Janus Specification (Figure 4) and allows users to specify their requirements in the form of a questionnaire. After specification, users are asked to assign a weight to each specified item in a *weighting sheet*.

The specified items are shown in the *specification* window in Figure 7. Clicking on an item provides users with physical necessary-condition rules (*specification-linking rules*) for a kitchen design to satisfy the item, as seen in the two lines in the middle of the *specification* window in Figure 7. Given this information, users can explore the arguments behind the rules. The rules shown on the screen are mouse-sensitive. Clicking on one of them activates Janus Argumentation, which provides more detailed information. Figure 5 illustrates the rationale behind the rule 'the stove should be away from a door if a user wants a kitchen to be safe.' Invoking the *retrieve from specification* command orders the design examples in terms of their *appropriateness* values to the specified items (see the *matching designs* window in Figure 7).

Users can now retrieve design examples that are similar to the current construction. When invoking the *retrieve from construction* command, they are asked to choose a criterion (a *parsing topic*) for defining the similarity between the current construction and the design examples in the catalogue. When they choose 'design unit types' as a parsing topic, a menu comes up that allows them to select all or some of the design unit types being used in the current construction. In Figure 8 a user has selected all the appliances that were used in the construction of Figure 3. The system then retrieves examples that contain the specified design unit types.

These interactions gradually narrow the catalogue space, providing users with a small set of examples that are relevant to the current construction and are ordered in terms of their appropriateness. Users can examine them one at a time. If no objects appropriate to the current task are found, the specification may be modified by selecting other answers in the specification sheet, or by changing the weights in the weighting sheet, or both. The *retrieval from specification* command then reorders the examples. Users may use the *retrieval from construction* command, and choose other criteria for defining the similarity. This will retrieve another set of examples. Finally, users may be interested in one of the presented catalogue examples, and may bring it into the *one of the matching design examples* window. They then go back to Janus Construction with the *resume construction* command. Janus Construction automatically shows the selected example in its *catalogue window* (see Figure 3). Users can refer to this example for new ideas on how to proceed with the current construction, or they can replace the current construction with the example they have found.



```
30 Matching Designs
one-wall-kitchen [illegible]
khaldouns-kitchen [illegible]
l-shaped-kitchen [illegible]
learning-example-1 [illegible]
u-shaped-kitchen [illegible]
kumiyos-kitchen [illegible]
learning-example-3 [illegible]
corridor-kitchen [illegible]
gerhards-kitchen [illegible]
island-corner [illegible]
island-kitchen [illegible]
davids-kitchen [illegible]
learning-example-4 [illegible]
scotts-kitchen [illegible]
jonathans-kitchen [illegible]
learning-example-2 [illegible]
andi-kitchen [illegible]
helga-kitchen [illegible]

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
 Jonathans-Kitchen
 Gerhards-Kitchen
 Kumiyos-Kitchen

One-Wall-Kitchen

Specification
 [0.6] Size of family? Small
 [0.3] Do both husband and wife work? Both
 [0.3] Who does the cooking? Wife
 [0.7] Cook's approximate height? 5'-5'6"
 [0.7] Right Handed or left handed? Left
 [0.8] How many meals are generally prepared a day?
 2
 [0.3] Do you usually use a dishwasher? No
 [1.0] Is safety important to you? Yes
  -Following conditions should be kept for this specifi
 cation:
      • STOVE is away from WINDOW.
      • STOVE is away from DOOR.
 [0.4] Are you interested in an efficient kitchen? Yes

 ▶ Show Item 'Kumiyos-Kitchen'
 ▶ Add To Bookmarks
 ▶ Show Item 'Gerhards-Kitchen'
 ▶ Add To Bookmarks
 ▶ Show Item 'Jonathans-Kitchen'
 ▶ Add To Bookmarks
 ▶ Retrieve From Specification
 ▶ Show Item 'One-Wall-Kitchen'
 ▶
 ▶ ▮
```

Fig. 7. A screen image of CatalogueExplorer.
The leftmost *Matching Designs* window lists all the currently retrieved design examples, ordered according to their appropriateness to the current specification. The *Bookmarks* window is used as a temporary name-holder of catalogue items. The two panes in the middle show one of the matching examples in detail (the top pane provides a set of slot values and the bottom pane provides a floor layout). The *Category Hierarchy* window shows the hierarchical structure of the catalogue. The *Specification* window shows specified items with the assigned weight of importance (the result of Figure 4). The items in this window are mouse-sensitive, and by clicking on one of them, CatalogueExplorer provides information about the corresponding *specification-linking rules* (the two lines in the middle of the window). Clicking on a rule activates Janus Argumentation, which provides the underlying argumentation for that rule (see Figure 5).

5.3. Retrieval mechanisms

5.3.1. Retrieval from specifications

To use a partial specification to identify a relevant design object, we must consider the following issues: types of specifications, and weighting importance for dealing with multiple contradictory features.

Types of specifications. There are two types of specifications for a design: *surface features* and *hidden features*. For example, the specification 'a kitchen with

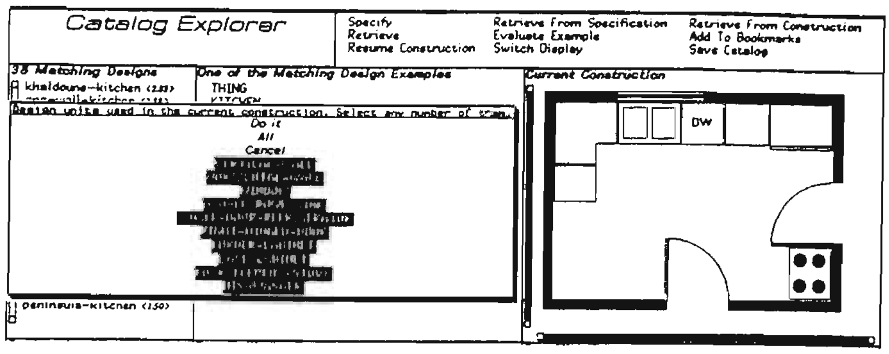


Fig. 8. Retrieve from Construction.
The *retrieve from construction* command with a *parsing topic* 'design unit types' analyses the current construction, and provides a list of all the design unit types being used in the construction. Users can select which design unit types they consider to be most important for locating prestored designs in the catalogue.

a dishwasher' is a surface feature that explicitly describes the design, whereas 'a kitchen suitable for small children' is a hidden feature of the design (it is not explicitly expressed in the final design (Kolodner, 1990)). Surface features are determined by the structure of a design, whereas hidden features are related to functions and behavior of the design (Gero, 1990). Hidden features can be computed or inferred only by using domain knowledge.

In practice, initial customer questionnaires given by professional kitchen designers to their customers often ask questions that relate to hidden features. The expertise, or domain knowledge, of the designers allows them to map these specifications into concrete structural features.

Surface features are represented in terms of a solution domain. In contrast, hidden features are often represented in terms of a problem domain. Mechanisms for retrieving design objects from specifications should, therefore, vary according to their type. Design examples can be retrieved from the catalogue by surface-feature specifications with a conventional query mechanism, because the surface features are already represented in the solution domain. In contrast, in order to retrieve design examples by specifying hidden features, the system must have the domain knowledge to relate these features to the solution structure.

Weighting importance. Sometimes specified items contradict each other. Users may not notice the contradictions if they occur between hidden features. If this happens, the system will be unable to retrieve design examples that satisfy the specifications, because (of course) such examples do not exist. Consider the specifications 'a safe kitchen' and 'a kitchen that provides easy access to the dining area.' Although they do not seem to contradict each other, they do so in terms of hidden features. As Figure 5 shows, a stove should be away from a door for the first specification, but close to a door for the second specification.

To resolve the conflict, users must prioritize the specifications and make trade-offs. They must indicate the importance of the specifications by assigning a weight to each specification. If they specify that 'a safe kitchen' is more important to them, the stove should be placed away from the doors.

Specification-linking rules. CatalogueExplorer automatically infers subjective hidden features of design examples in the catalogue by using domain knowledge in the form of *specification-linking rules* (see Figure 9). The *specification-linking rules* link each subjective hidden-feature specification to a set of physical-condition rules. For example, in the middle of the *specification* window in Figure 7, two rules are shown ('stove is away from door' and 'stove is away from window'). These are conditions for a kitchen to have the hidden feature 'a safe kitchen' (Figure 4).

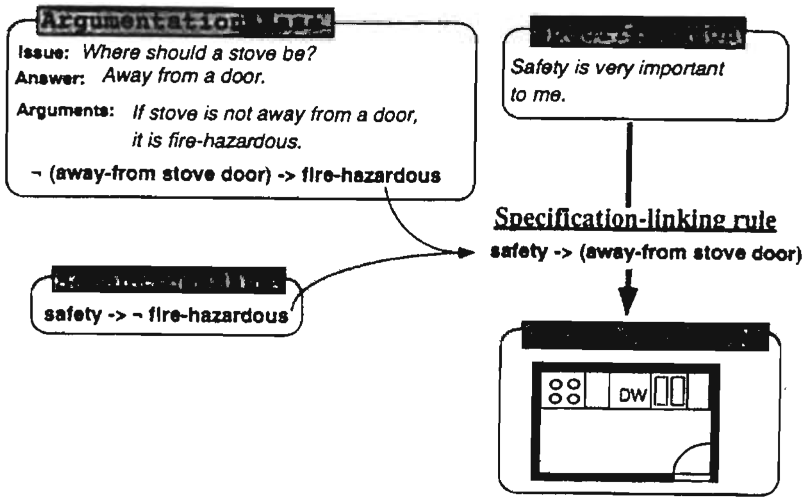

Argumentation
Issue: Where should a stove be?
Answer: Away from a door.
Arguments: If stove is not away from a door, it is fire-hazardous.
¬ (away-from stove door) -> fire-hazardous
Safety is very important to me.
Specification-linking rule
safety -> (away-from stove door)
safety -> ¬ fire-hazardous
DW

Fig. 9. Specification-linking rules in CatalogueExplorer.

The important aspect of the *specification-linking rules* is that they can be dynamically derived from the content of Janus Argumentation (see Figure 9). Suppose that the system has the following internal representation for the *fire hazard* argument shown in Figure 5:

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

and the system has the domain knowledge:

$$\text{SAFETY} \rightarrow \sim \text{FIRE-HAZARDOUS} \qquad (2)$$

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

$$\sim \text{FIRE-HAZARDOUS} \rightarrow (\text{Away-from-p STOVE DOOR}) \qquad (3)$$

From (2) and (3) we get:

$$\text{SAFETY} \rightarrow (\text{Away-from-p STOVE DOOR}) \qquad (4)$$

5.3.2. Retrieval from construction

For the retrieval of design examples that are relevant to a partial construction, we must deal with the issues of matching design examples in terms of the surface features of a design, i.e. at the structural level. The issues discussed in the previous section, such as partial matching and factor of importance, also hold here.

Domain-specific parsers analyse the design under construction. They represent the user's criteria for the articulation of the task at hand from a partial construction. That is, they determine how similarities between the partial construction and a design example in the catalogue are to be defined for the retrieval of design examples from the catalogue (see Figure 8).

CatalogueExplorer supports the following two parsers (users have a mechanism for choosing 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 analyses the current construction, finds which design unit types are used, and provides the user with a menu to select some of them (see Figure 8).
- *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 the sink, examples that match this configuration are retrieved.

5.4. Discussion of CatalogueExplorer

In CatalogueExplorer, users gradually narrow a catalogue space. The system can dynamically infer hidden features of catalogue examples, and provide users with an explanation of the inference mechanism. The system retrieves examples that are similar to the current construction, and provides users with directions in which to proceed; or it warns them of potential failures. The retrieved information may remind them of ideas which they had not thought of before, thus inspiring them to develop creative solutions to their problems.

By using the environment over time, catalogue examples are collected incrementally. The system allows designers to store design examples in the catalogue (currently 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 designers to access good examples easily and prevent a chaotic growth in the size of the catalogue. However, by not including failure cases, they do not help designers to discover what went wrong in the past.

Our design environments empower both inexperienced and experienced designers. The system is useful for inexperienced designers because it supports learning on demand (Fischer, 1991). It is useful for experienced designers because it allows them to incrementally accumulate domain knowledge into the system. Interactions with numerous 'experts' have led us to believe that expert knowledge is never complete, because design situations are idiosyncratic and unique.

A major limitation of the current system is the relatively small size of the catalogue, which contains less than 100 examples. Many problems of effectively managing large information spaces have therefore not been dealt with. However, the authors are concerned about the limited cognitive resources of humans, and are not greatly concerned about computational resources. Because there are no mechanisms for associating formal representations with arguments, the *specification-linking rules* must be manually derived. The parsers for analysing partial constructions need to be extended to deal with more abstract levels, such as an emerging shape (e.g. an L shape or a U shape). Currently, these have to be specified by the user. A combinatorial use of the structural features for detecting emerging features should be explored, such as the connectionist approach described by Newton and Coyne (1991).

6. Amplifying human creativity with computers

We are interested in human creative potential—not just with analysing it, but with asking *how people can become more creative*. We are convinced that the power of the unaided mind is highly overrated, and that much of human intelligence and creativity results from our technology (Norman, 1993). Knowledge in the head needs to be augmented by knowledge in the world. However, large quantities of information do not necessarily enhance creative design, or problem solving, or decision making: in fact they may overwhelm people with too much information. The challenge is to say the 'right' thing at the 'right' time (Fischer, Nakakoji et al., 1992).

Our approach is to build domain-oriented design environments to empower people, rather than to build expert systems to replace them (Fischer, 1990; Fischer and Nakakoji, 1991). These environments aim to inform and support the judgment of designers, rather than 'de-skilling' them by judging for them, or designing for them. Designers who use these systems are free to ignore, or turn off, or alter the critiques that the system provides. We have pursued this approach not only because automative approaches have failed in many domains (e.g. software design (Barstow, 1983) and machine translation (Kay, 1980)), and not only because serious doubts have been articulated about the 'in principle' limitations of expert systems

(Winograd and Flores, 1986). We have also pursued them because we believe that *people enjoy 'doing' and 'deciding'*. People enjoy the process, not just the final product. They wish to participate. This is why they build model trains, plan their vacations, and design their own kitchens.

Building cooperative problem-solving systems allows us to exploit the relative strengths of the two participants: people are creative and can put tasks into larger contexts; computers are effective repositories and managers of large amounts of information. We have chosen design (Simon, 1981) as the domain in which to explore issues in creativity. Design incorporates many cognitive issues, such as recognizing and framing a problem, understanding given information, adapting generic information to the idiosyncrasies of a situation, and relating partial specifications and partial constructions to a catalogue of prestored designs.

Our research is based on the conceptual framework we have outlined: the integration of action, assessment, and reflection. By engaging in reflection in action with the use of computational environments, we have created situations that 'talk back' to us. Our system-building efforts, and the use of these systems, create breakdowns, which trigger further reflection. This has given rise to a large number of issues that need to be addressed in the future:

- Are there differences in the performance, quality, and creativeness of the product if the system is used with or without critics, the catalogue and the simulation components?
- What are the tradeoffs between running the system in a critiquing versus a constraint mode (Gross and Boyd, 1991), where the latter prevents certain problems from arising (e.g. by enforcing building codes), and the former provides designers with opportunities to deal with breakdowns?
- What are the tradeoffs between different intervention strategies, e.g. between displaying enough information as opposed to disrupting the work process? When are designers willing to suspend the construction process in order to access relevant information? Does 'making information relevant to the task at hand' prevent serendipity (Roberts, 1989)?
- If an environment can always supply the information that the situation requires, why would users bother to learn the information (Fischer, 1991)?
- Under what conditions will designers challenge or extend the knowledge represented in the system? How can they be motivated to do so (Fischer, Girgensohn et al., 1992)?
- Should the 'back-talk' be embedded directly into the artifact, or should it be handled by a separate discourse (such as feedback from the critiquing and simulation components)?
- To what extent are situations and reflective conversations controlled by media properties?
- How can a balance be achieved between technical rationality (e.g. the use of plans and rules) and reflective action (Ehn, 1988; Suchman, 1987)? Even if it is true that 'design is more than the application of standard principles,' it does

not follow that principles are not useful.

We hope that our ongoing research efforts in designing, building and evaluating design environments will increase the ability of these environments to amplify human creativity.

Acknowledgements

The authors wish to thank the members of the Human-Computer Communication group at the University of Colorado, who contributed substantially to the conceptual framework and the systems discussed in this paper. The research was supported by the US National Science Foundation under Grants IRI-9015441 and MDR-9253245, by Software Research Associates, Inc. (Tokyo, Japan), and by NYNEX Science and Technology Center (White Plains, N. Y.).

References

Barstow, D.: 1983, A perspective on automatic programming, *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, pp. 1170–1179.

Boden, M.: 1990, *The Creative Mind: Myths & Mechanisms*, Weidenfeld and Nicolson, London.

Conklin, J., and Begeman, M.: 1988, gIBIS: A hypertext tool for exploratory policy discussion, *Proceedings of the Conference on Computer Supported Cooperative Work*, ACM, New York, pp. 140–152.

Ehn, P.: 1988, *Work-Oriented Design of Computer Artifacts*, Almquist & Wiksell International, Stockholm.

Fischer, G.: 1989, Creativity enhancing design environments, *Proceedings of the International Conference on 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*, **15**: 1, pp. 21–36.

Fischer, G.: 1991, Supporting learning on demand with design environments, *Proceedings of the International Conference on the Learning Sciences*, Evanston, IL, pp. 165–172.

Fischer, G., Böcker, H. D.: 1983, The nature of design processes and how computer systems can support them, Degano, P., Sandewall, E. (eds) *Proceedings of the European Conference on Integrated Interactive Computer Systems*, North Holland, pp. 73–88.

Fischer, G., Girgensohn, A.: 1990, End-user modifiability in design environments, *Human Factors in Computing Systems, CHI'90 Conference Proceedings*, ACM, New York, pp. 183–191.

Fischer, G., Girgensohn, A., Nakakoji, K., Redmiles, D.: 1992, Supporting software designers with integrated, domain-oriented design environments, *IEEE Transactions on Software Engineering*, **18**: 6, pp. 511–522.

Fischer, G., Grudin, J., Lemke, A. C., McCall, R., Ostwald, J., Reeves, B. N., Shipman, F.: 1992, Supporting indirect, collaborative design with integrated knowledge-based design environments, *Human Computer Interaction*, **7**: 3, pp. 281–314.

Fischer, G., Henninger, S., Nakakoji, K.: 1992, DART: Integrating information delivery and access mechanisms. Unpublished Manuscript, Department of Computer Science, University of Colorado, Boulder.

Fischer, G., Lemke, A. C.: 1988, Construction kits and design environments: steps toward human problem-domain communication, *Human-Computer Interaction*, **3**: 3, pp. 179–222.

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

Fischer, G., Lemke, A. C., McCall, R., Morch, A.: 1991, Making argumentation serve design, *Human Computer Interaction*, **6**: 3-4, pp. 393–419.

Fischer, G., McCall, R., Morch, A.: 1989, JANUS: Integrating hypertext with a knowledge-based design environment, *Proceedings of Hypertext'89*, ACM, pp. 105–117.

Fischer, G., Nakakoji, K.: 1991, Making design objects relevant to the task at hand, *Proceedings of AAAI91, Ninth National Conference on Artificial Intelligence*, AAAI Press/MIT Press, Cambridge, MA, pp. 67–73.

Fischer G., Nakakoji, K., Ostwald, J., Stahl, G., Sumner, T.: 1993, Embedding critics in integrated design environments, Knowledge Engineering Review Journal, Cambridge University Press, **4**: 8, pp. 285–307.

Fischer, G., Nieper-Lemke, H.: 1989, HELGON: Extending the retrieval by reformulation paradigm, *Human Factors in Computing Systems, CHI'89 Conference Proceedings*, ACM, New York, pp. 357–362.

Fischer, G., Reeves, B. N.: 1992, Beyond intelligent interfaces: exploring, analyzing and creating success models of cooperative problem solving, *Applied Intelligence*, **1**, pp. 311–332.

Gero, J. S.: 1990, A locus for knowledge-based systems in CAAD education, *in* McCullough, M., et al. (eds), *The Electronic Design Studio*, MIT Press, Cambridge, MA, pp. 49–60.

Gross, M. D., Boyd, C.: 1991, Constraints and knowledge acquisition in Janus, *Technical Report, Department of Computer Science, University of Colorado, Boulder.*

Halasz, F. G.: 1988: Reflections on NoteCards: seven issues for the next generation of hypermedia systems, *Communications of the ACM*, **31**: 7, pp. 836–852.

Henderson, A., Kyng, M.: 1991, There's no place like home: continuing design in use, *in* Greenbaum, J., Kyng, M. (eds), *Design at Work: Cooperative Design of Computer Systems*, Lawrence Erlbaum, Hillsdale, NJ, pp. 219–240.

Hutchins, E. L., Hollan, J. D., Norman, D. A.: 1986, Direct manipulation interfaces, *in* Norman, D. A., Draper, S. W. (eds), *User Centered System Design, New Perspectives on Human-Computer Interaction*, Lawrence Erlbaum, Hillsdale, NJ, pp. 87–124.

Jacob, F.: 1977, Evolution and tinkering, *Science*, **196**: 4295, pp. 1161–1166.

Kay, M.: 1980, The proper place of men and machines in language translation, *Technical Report CSL–80–11, Xerox Palo Alto Research Center.*

Kolodner, J. L.: 1990, What is case–based reasoning? *in* AAAI90 Tutorial on Case-Based Reasoning, pp. 1–32.

Lave, J.: 1988, *Cognition in Practice*, Cambridge University Press.

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., Fischer, G.: 1990, A cooperative problem solving system for user interface design, *Proceedings of AAAI90, Eighth National Conference on Artificial Intelligence*, AAAI Press/MIT Press, Cambridge, MA, pp. 479–484.

McCall, R.: 1986, Issue-Serve Systems: a descriptive theory of design, *Design Methods and Theories*, **20**: 8, pp. 443–458.

McLaughlin, S., Gero, J. S.: 1989, Creative processes: can they be automated? Reprints of the International Conference on Modeling Creativity and Knowledge-Based Creative Design, Heron Island, Australia, pp. 69–94.

Newton, S., Coyne, R. D.: 1991, The impact of connectionist systems on design, *in* Gero, J. (ed.), *Artificial Intelligence in Design'91*, Butterworth Heinemann, Oxford, pp. 49–75.

Nielsen, J., Richards, J. T.: 1989, The experience of learning and using Smalltalk, *IEEE Software* pp. 73–77.

Norman, D. A.: 1993, *Things That Make Us Smart*, Addison Wesley, Reading, MA.

Owen, D.: 1986, Answers first, then questions, *in* Norman, D. A., Draper, S. W. (eds), *User Centered System Design, New Perspectives on Human-Computer Interaction*, Lawrence Erlbaum, Hillsdale, NJ, pp. 361–375.

Patel-Schneider, P. F.: 1984, Small can be beautiful in knowledge representation, *AI Technical Report 37, Schlumberger Palo Alto Research Center*.

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

Riesbeck, C. K.: 1988, An interface for case-based knowledge acquisition, *in* Kolodner, J. (ed.), *Proceedings: Case-Based Reasoning Workshop*, Morgan Kaufmann, Clearwater Beach, FL, pp. 312–326.

Rissland, E. L., Skalak, D. B.: 1989, Combining case-based and rule-based reasoning: a heuristic approach, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, Palo Alto, pp. 524–530.

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

Roberts, R. M.: 1989, *Serendipity: Accidental Discoveries in Science*, John Wiley & Sons, New York.

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

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

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

Williams, M. D.: 1984, What makes RABBIT run? *International Journal of Man-Machine Studies*, **21**, pp. 333–352.

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