Beyond the macho approach of artificial intelligence: empower human designers — do not replace them

G Fischer and K Nakakoji*

Designers deal with ill defined and wicked problems that are characterized by fluctuating and conflicting requirements. Traditional design methodologies that are based on the separation between problem setting (analysis) and problem solving (synthesis) are inadequate for the solution of these problems. The supporting of design with computers requires a cooperative problem-solving approach that empowers designers with integrated, domain-oriented, knowledge-based design environments.

The paper describes the motivation for the latter approach, and introduces an architecture for such design environments. It focuses on the integration of specification, construction, and a catalogue of prestored design objects in those environments to illustrate how such integrated design environments empower human designers. The CatalogExplorer system component, which is described in detail, assists designers in the location of examples in the catalogue that are relevant to the task at hand, as partially articulated by the current specification and construction. Users are thereby relieved of the tasks of forming queries or navigating in information spaces. The last part of the paper discusses the relationship of the work with the conceptual framework developed by Donald Schön.

Keywords: integrated, domain-oriented, knowledge-based design environment, design-support systems, coevolution of problem setting and problem solving, relevance to the task at hand, reflection in action, multifaceted architecture, user interfaces for design, computer-supported cooperative work

Department of Computer Science and Institute of Cognitive Science, University of Colorado, Boulder, Colorado 80309–0430, USA
*At the above address, and at the Software Engineering Laboratory, Software Research Associates Inc., 1–1–1 Hirakawa-cho, Chiyoda-ku, Tokyo 102, Japan
The paper is a substantially revised and extended version of the paper that appeared in **Gero, J (Ed.)** *Artificial Intelligence in Design* Butterworth–Heinemann, UK (1991) (*Proc. Artificial Intelligence in Design '91 Conf.* Edinburgh, UK (25–27 Jun 1991)).
Revised paper received 9 September 1991. Accepted 24 October 1991

Design is an ill defined[1] or wicked[2] problem that has fluctuating and conflicting requirements. Early design methods, which are based on directionality, causality, and the separation of analysis from synthesis, are inadequate for the solution of such problems[3].

The research effort discussed in this paper is based on the assumption that these design problems are best solved by the support of a cooperative problem-solving approach between humans and integrated, domain-oriented, knowledge-based design environments[4]. The combination of knowledge-based systems and innovative human–computer communications techniques empowers humans to produce 'better' products by augmenting their intellectual capabilities and productivity, rather than simply by the use of an automated system[5].

The authors' approach is not that of building another expert system. Expert systems require a rather complete understanding of a problem to start with. This is an assumption that does not hold for ill defined problems. For a set of rules to be produced for an expert system, the relevant factors and the background knowledge need to be identified. However, this information cannot be fully articulated. What has been made explicit always sets a limit, and there exists the potential for breakdowns that require a move beyond this limit[6].

This paper uses the domain of the architectural design of kitchen floor plans as an 'object to think with', for the purposes of illustration. The simplicity of the domain helps in concentration on the essential issues of the approach, without distraction by understanding of the semantics of the domain itself. General issues of design

 0950–7051/92/010015–16

environments are first discussed, with emphasis on the importance of domain orientation and integration of these environments. Then, the *multifaceted architecture* that underlies these environments is described. This serves as a conceptual framework for the research. These environments support three important design concepts: (a) reflection in action, (b) the evolution of individual design projects and design environments, and (c) information being made relevant to the task at hand. CatalogExplorer, an innovative system component, illustrates how the integrated environment empowers human designers in terms of the third notion. CatalogExplorer integrates specification, construction, and a catalogue of prestored design objects. The synergy of this 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 tasks of forming queries or navigating in information spaces for retrieval. Related work, and the achievements and limitations of the system, are briefly described. The last part of the paper discusses the relationship of the work to the conceptual framework developed by Donald Schön[7,8].

PROBLEMS

Integration of problem setting and problem solving

The integration of problem setting and problem solving is indispensable for design problems that are characterized as ill defined problems[7]. As Simon mentioned[9], complex designs are implemented over a long period of time, and they 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 having information about it. Professional practitioners have at least as much to do with the definition of the problem as with the solution of the problem[2].

An empirical study by the authors' research group, which analysed *human–human cooperative problem solving* between customers and sales agents in a large hardware store[10], 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, on the basis of the feedback that they get from their environment.

In designing, this feedback is provided by the 'back talk of the situation'[7]. While engaging in a 'conversation with the design material', designers become aware of an occurrence of a breakdown. This awareness is triggered by evaluation and appreciation of the current design stage (artefact) in terms of their task at hand (goal). The evaluation is carried out either by the designers themselves, or by outside agents, such as design teachers or computational agents (such as critics), in design environments. This reflection of the action results in the determination of a next move in problem setting and in problem solving.

The integration of problem setting (analysis) and problem solving (synthesis) is not supported in first-generation design methodologies or in traditional approaches to software design[11]. Automated design methodologies fail, because they need a complete requirement specification to be established before design is started.

Domain orientation

While computers are regarded as a design medium, it is necessary to reduce the great transformation distance between a design substrate and an application domain[12]. Designers should perceive design as communication with an application domain, rather than as the manipulation of symbols on computer displays. The computer should become invisible by supporting *human problem-domain communication*, and not just human–computer communications[13]. Human problem-domain communication provides a new level of quality in human–computer communications 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 artefacts from applications-oriented building blocks of various levels of abstraction, according to the principles of the domain.

Retrieval of information relevant to task at hand

In the support of the integration of problem setting and problem solving in design environments, the identification of information that is relevant to the task at hand is crucial. Every step made by a designer towards a solution determines a new space of related information, which cannot be determined *a priori*, owing to its very nature. Integrated design environments are based on high-functionality systems[14] that contain 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 systems support, it is difficult to locate and understand the objects[15,16].

The task at hand is usually represented in terms of a problem domain rather than a solution domain. This leads to the inapplicability of conventional database retrieval techniques[17], which require humans to articulate what they are looking for by formulating a highly specific query in terms of a solution domain. For example, suppose that a novice designer wants to design a floor plan for a safe kitchen. 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 task, namely useful floor plans for the design of a safe kitchen (see Figure 1). If users can articulate what they need, a query-based search can lessen the burden of the location of promising objects[18].

With *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[19]. Navigational access requires the information space to have 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 the identification of a small set of relevant information.

Design environments need other mechanisms (as discussed in this paper) that can identify small sets of

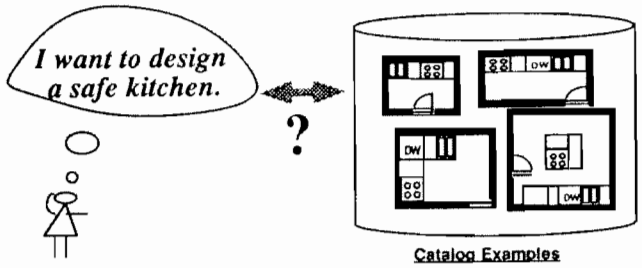


Figure 1. Location of relevant information to task at hand
[There is no clue to help the designer to access information in the catalogue (*solution domain*) that is relevant to the task at hand, i.e. what the designer has in mind when requesting a safe kitchen (*problem domain*).]

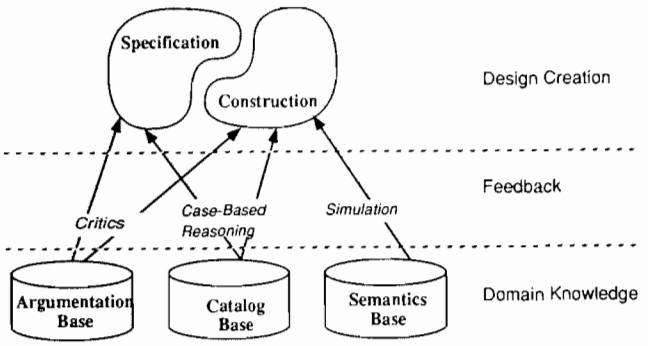


Figure 2. Elements of multifaceted architecture

objects that are relevant to the task at hand. Users must be able to articulate incrementally the task at hand. The information provided in response to these problem-solving activities based on partial specifications and constructions must assist users to refine the definition of their problem.

MULTIFACETED ARCHITECTURE FOR INTEGRATED DESIGN ENVIRONMENTS

During 1987–1991, several prototype systems of domain-oriented design environments[20,21] have been developed and evaluated. These different system-building efforts help in the definition of a multifaceted architecture, which consists of five elements (see Figure 2): (a) specification, (b) construction in support of design creation, (c) an argumentation base, (d) a catalogue base, and (e) a semantic base, as information depositories for domain knowledge. Figure 3 shows the interface components of the multifaceted architecture from a user's point of view. These components are described below 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 it runs on Symbolics LISP machines. Currently, Janus consists of four major subsystems: Janus Construction, Janus Argumentation, CatalogExplorer and Janus Modifier[16]. Each subsystem supports different aspects of the architecture.

Although the importance of domain orientation has been emphasized, this architecture should not be regarded as a specific framework for a certain domain. On the contrary, it is assumed that the architecture presented in this paper serves as a generic framework for the construction of a class of domain-specific environments.

Components of multifaceted architecture

Integrated design environments that are based on the multifaceted architecture are composed of the following five interface components (see Figure 3):

- A *construction kit* is the principal medium for the implementation of design. It provides a palette of domain abstractions, and it supports the construction of artefacts, using direct manipulation and other interaction styles. A construction represents a concrete implementation of a design, and it reflects a user's current problem situation. Figure 4 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[22] (IBIS), and they are linked according to what information resolves an issue that is relevant to a partial construction. The issues, answers and arguments held in Janus Argumentation (see Figure 5) can be accessed via links from the domain knowledge in other components.
- A *catalogue* (see Figures 4 and 6) provides a collection of prestored design objects that illustrates the space of possible designs in the domain. Catalogue examples support reuse and case-based reasoning[23,24].
- A *specification component* (see Figure 7) allows designers to describe some required characteristics of the design at a high level of abstraction, and it assigns weights of the importance to each specified item. The specifications are expected to be modified and augmented during the whole design process, rather than to be fully articulated before the design is started. They are used to prioritize all the information spaces in the system with respect to the emerging task at hand.
- A *simulation component* allows 'what-if' games to be carried out to allow designers to simulate usage scenarios with the artefact that is being designed. Simulation complements the argumentative component.

Integration in multifaceted architecture

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, a synergistic whole being formed. Links between the components of the architecture are supported by various mechanisms (see Figure 3). The major mechanisms included are listed below.

- The Construction Analyzer is a critiquing component[25] 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[6], 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 4 and 5).

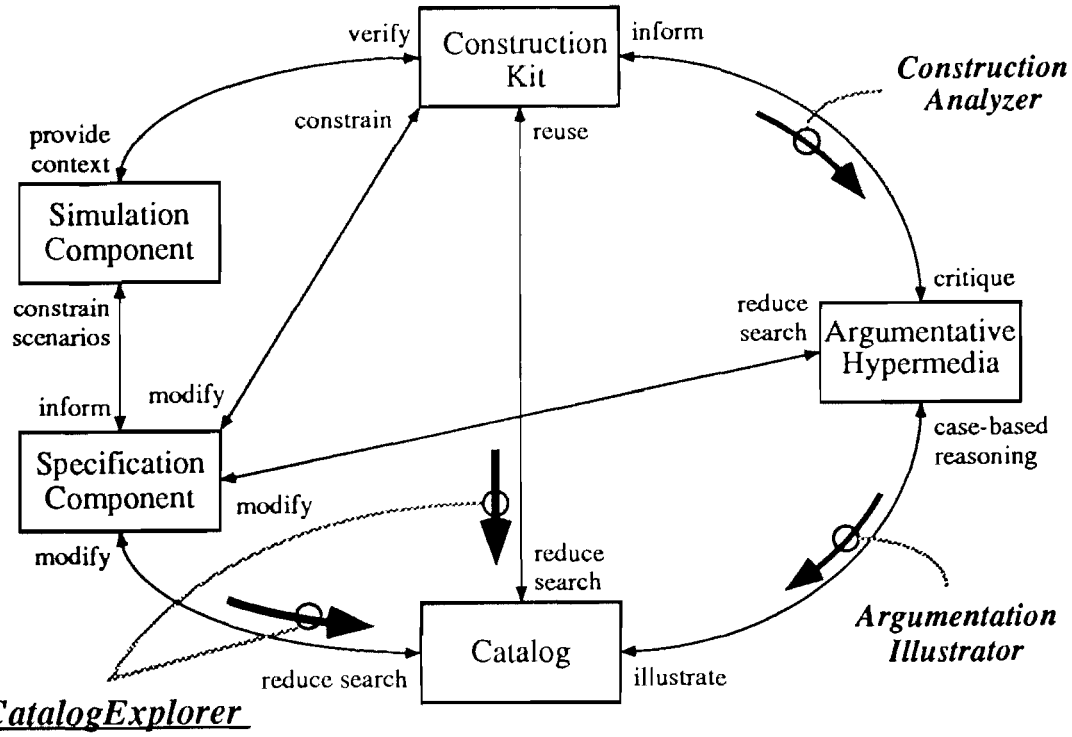


Figure 3. Interface components for multifaceted architecture
[Support for links between the components is crucial for synergy of integration.]

- The Argumentation Illustrator helps users to understand the information given in an argumentative hypermedium by using a catalogue design example as a source of concrete realization (see Figure 5). 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 below in detail, helps users to search the catalogue space according to the task at hand. It retrieves design examples that are similar to the current construction situation, and it orders a set of design examples by their appropriateness to the current specification.

Design within multifaceted architecture

Design environments based on the multifaceted architecture support the following three design activities:

- *Reflection in action:* Design (as supported by the multifaceted architecture) iterates through cycles of specification, construction, evaluation and reuse in the working context. At each stage in the design process, the partial design that is embedded in the design environment is a stimulus that suggests what users should attend to next. The direction to new subgoals permits new information to be extracted from memory and reference sources, and it leads to new steps towards the development of the design. The integration of various aspects of design enables the situation to 'talk back' to users[7], following the characterization of design activities by Schön (the terms in square brackets are the authors' annotations):

> 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].

The relationship of the authors' approach to that of Schön is further discussed below.

- *Evolution of individual design projects and design environments:* Figure 8 shows the coevolution of specification and construction in an environment that is based on the multifaceted architecture. A typical cycle of events in the environment includes the following: (a) designers create a partial specification or a partial construction, (b) they do not know how to continue with this process, and so (c) they switch and consult other components in the system that provide them with information that is relevant to the partially articulated task at hand, and (d) they are able to refine their understanding on the basis of the *back talk* of the situation. As designers go back and forth between the components, the problem space is narrowed, and different facets of the artefact are refined. A completed design artefact (consisting of a specification and a construction) may be stored in the catalogue for later reuse. Through this process, the environment gradually evolves by being constantly used.
- *Articulation of the task at hand:* The integration enables the system to understand incrementally the task at hand. Suppose that a user is designing a kitchen as shown in Figure 9. In this example, the partially articulated task at hand is the determination of the location of a dishwasher in the given construction so that the kitchen will be safe and good for a left-handed person who has a small child. On the basis of this articulation, the system provides the user with relevant information without forcing the user to form

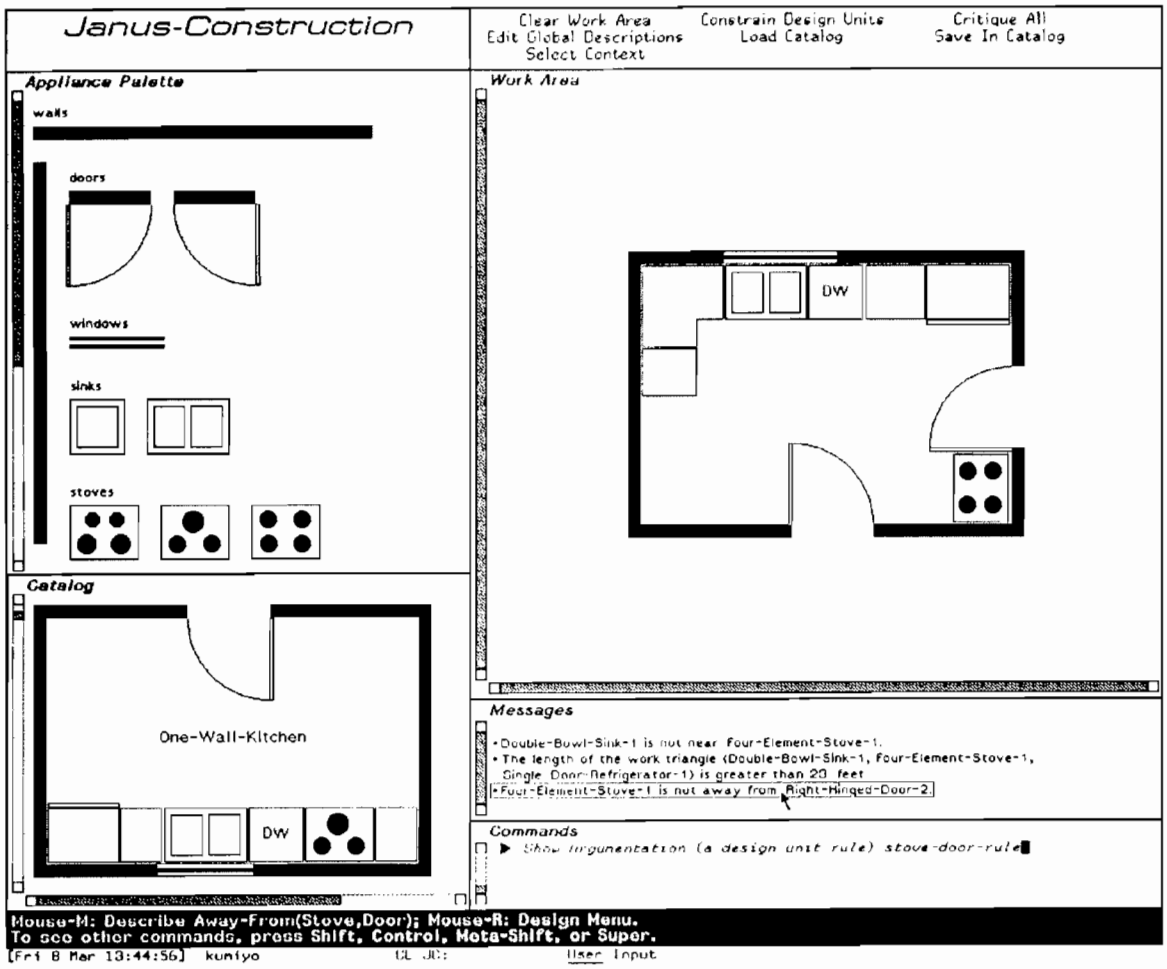


Figure 4. Janus Construction
[Building blocks (design units) are selected from the *palette*, and moved to desired locations inside the *work area*. Designers can reuse and redesign complete floor plans from the *catalogue*. The *messages* pane displays critiques automatically after each design change that triggers such a critic message (carried out by the Construction Analyzer). Clicking with the mouse on a message activates Janus Argumentation, and displays the argumentation related to that message (see Figure 5).]

queries or navigate through large information spaces to locate relevant information. This process is described in more detail below. By retrieving information in the same environment, the system can analyse usage patterns of the retrieved information and use them for refining the retrieval.

CATALOGEXPLORER

This section of the paper describes CatalogExplorer, which links the specification and construction components with the catalogue (see Figure 3), followed by a scenario that illustrates a typical use of the system. Then, the underlying mechanisms used in the scenario are described in more detail, with the mechanisms of retrieval from specification and retrieval from construction.

System description

Design objects stored in a catalogue can be used for (a) providing a solution to a new problem, (b) warning of possible failures, and (c) evaluating and justifying the decision[23,26]. The catalogue provides a source for different ideas in the same way as do the commercial catalogues that are shown by a professional kitchen designer to customers to help them understand their needs and make decisions. For large catalogues, the identification of design examples that are relevant to the task at hand becomes a challenging and time-consuming task (see Figure 1).

By integrating specification, construction and a catalogue, CatalogExplorer helps users to retrieve information that is relevant to the task at hand, and, as a result, it helps users to refine their partial specification and partial construction. Users need not form queries or navigate in a catalogue space to retrieve design objects from a catalogue, because their task at hand is partially articulated by a partial specification and construction.

The design examples in the catalogue are stored as objects in the Kandor[27] knowledge base. 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. Each design example can be (a) critiqued and praised by the Construction Analyzer, and (b) marked with a bookmark, which provides users with control in the selection of design examples and the forming of a task-specific small subset of the catalogue.

CatalogExplorer is based on the Helgon system[28], which instantiates the retrieval-by-reformulation paradigm[29]. It allows users to improve incrementally a query by critiquing the results of previous queries. Reformula-

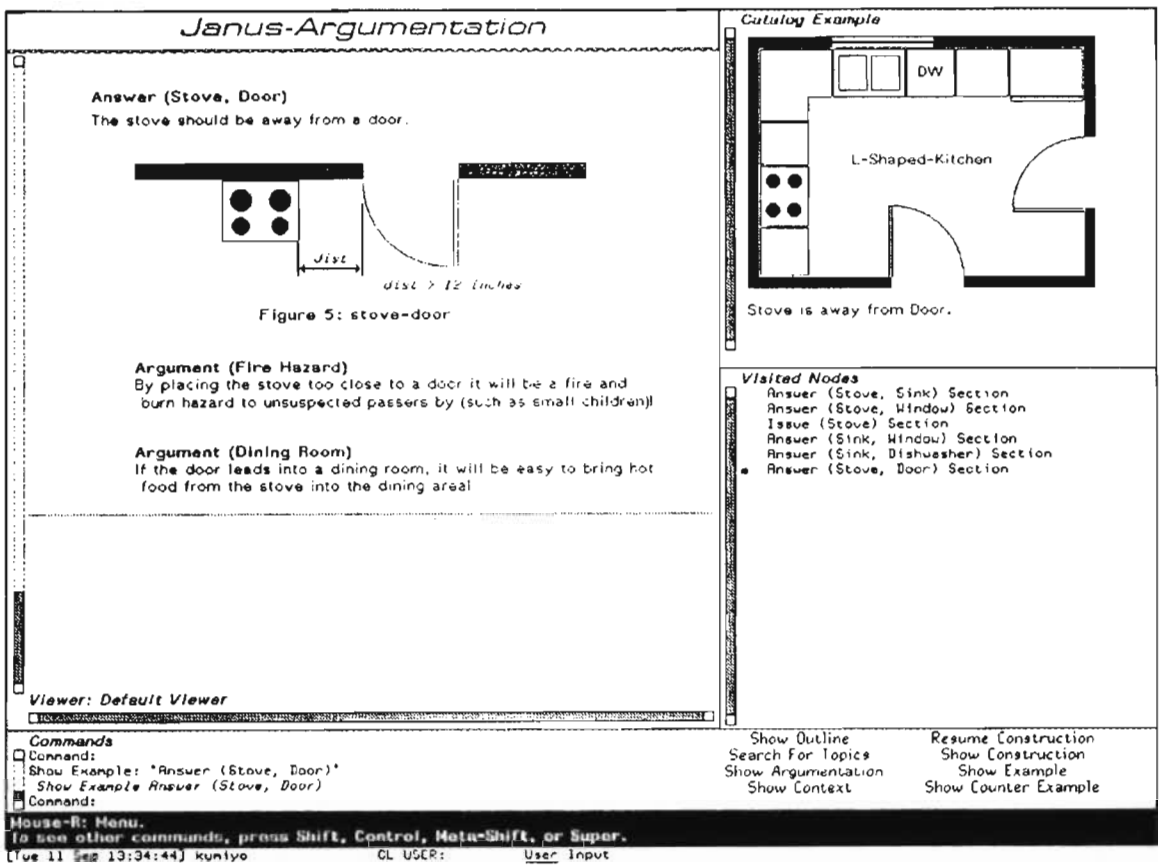


Figure 5. Janus Argumentation

[The screen image shows an answer to the question of where to locate the kitchen stove with respect to a door, and it 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-hand corner (which is activated by the 'show example' command in the *commands* pane) contextualizes an argumentative principle in relation to a specific design (done by the Argumentation Illustrator).

tion allows users to search iteratively for more appropriate design information, and to refine their specification, rather than be constrained to their initial specified query[17].

On the basis of the retrieval-by-reformulation paradigm, CatalogExplorer retrieves design objects that are relevant to the task at hand by using the following mechanisms:

- It exploits the information articulated in a partial specification to prioritize the designs stored in the catalogue (*retrieval from specification*).
- It analyses the current construction, and retrieves *similar* examples from the catalogue using similarity metrics (*retrieval from construction*).

Scenario with CatalogExplorer

CatalogExplorer (see Figure 6) is invoked by the *catalogue* command from Janus Construction (see Figure 4). The *specify* command provides a *specification sheet* (see Figure 7a) in the form of a questionnaire. After specification, users are asked to assign a weight to each specified item in a *weighting sheet* (see Figure 7b).

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 activates Janus Argumentation, providing 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'. By the *retrieve from specification* command being invoked, the design examples of the catalogue are ordered (see the *matching designs* window in Figure 6) by *appropriateness* values to the specified items.

Users can then retrieve design examples that are similar to their current construction. When invoking the *retrieve from construction* command, users are asked to choose a criterion (*parsing topic*) for the definition of the similarity between the current construction and design examples in the catalogue. When users choose 'design unit types' as a parsing topic, a menu comes up, as shown in Figure 10, that allows the user to select all or some of the design unit types being used in the current construction. In Figure 10, a user has selected all the appliances that were used in the construction of Figure 4. The system then retrieves examples that contain the specified design unit types.

The above interactions gradually narrow the catalogue

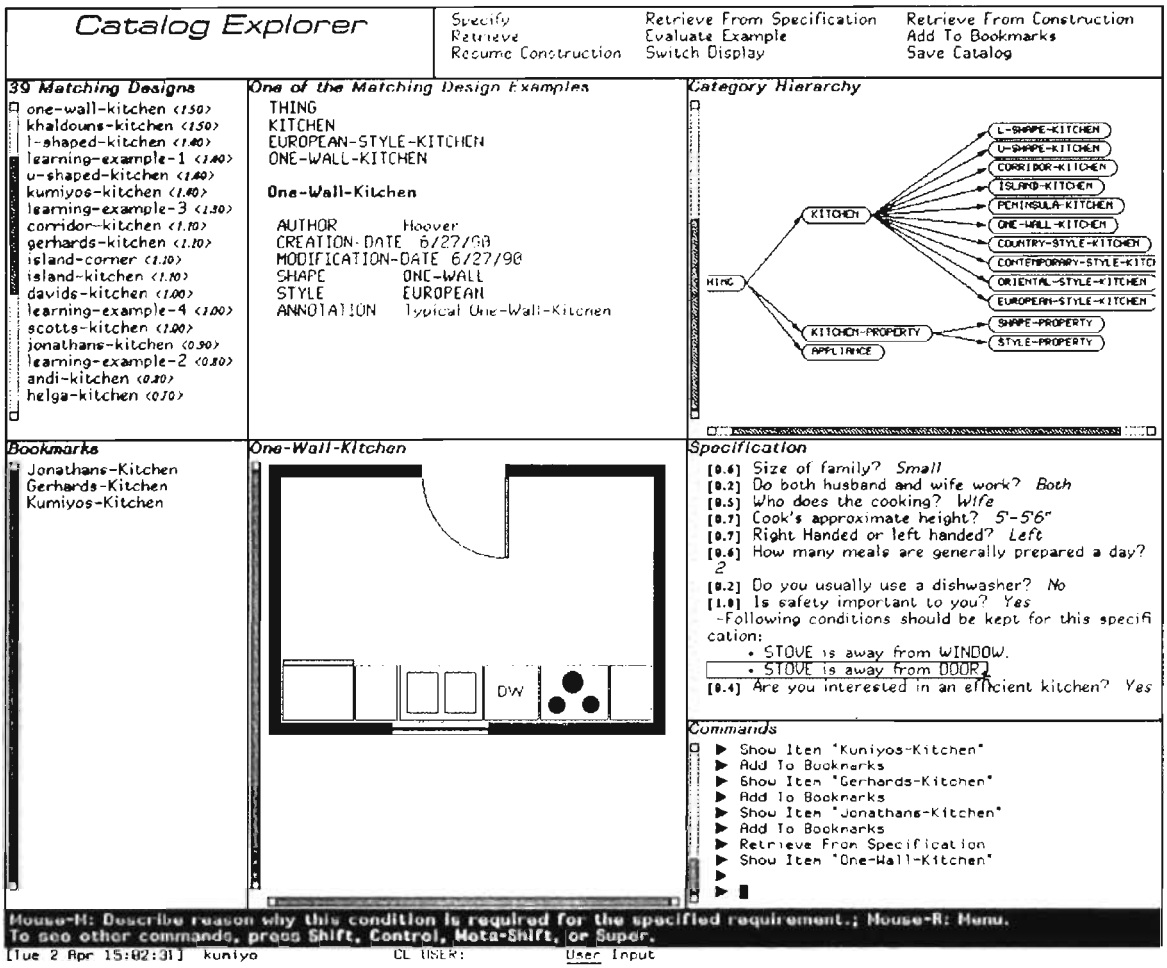


Figure 6. CatalogExplorer

[The leftmost *matching designs* window lists all the currently retrieved design examples in the catalogue, 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 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 7). The items in this window are mouse-sensitive, and if one is clicked on, CatalogExplorer provides the information of the corresponding *specification-linking rules* (two lines in the middle of the window). Clicking on one of the rules activates Janus Argumentation, which provides the underlying argumentation for that rule (see Figure 5).]

space, providing users with a small set of examples that are relevant to the current construction and ordered by the appropriateness to their specification. Users can examine them one by one with a reasonable amount of effort. If no objects that are appropriate to the current task are found, users may 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 reorders the examples. Users may use the *retrieval from construction* command, and choose other criteria for defining the similarity, which will retrieve another set of examples. Finally, they may decide which example they want to use by bringing the example into the *one of the matching design examples* window, and go back to Janus Construction with the *resume construction* command. Janus Construction automatically shows the selected example in the *catalogue* window of Janus Construction (see Figure 4). Users can refer to this example to get new ideas on how to proceed with their constructions, or they may replace the current construction with the example found.

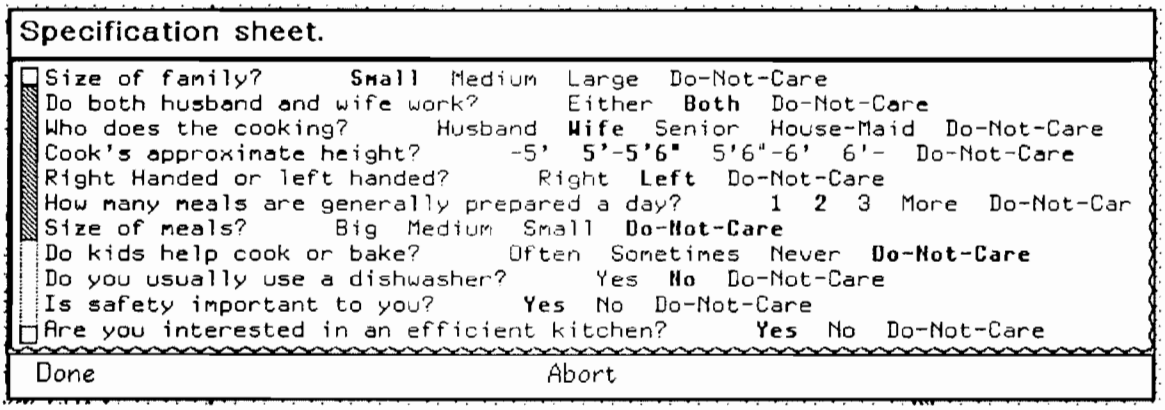
RETRIEVAL FROM SPECIFICATION

Issues related to specification

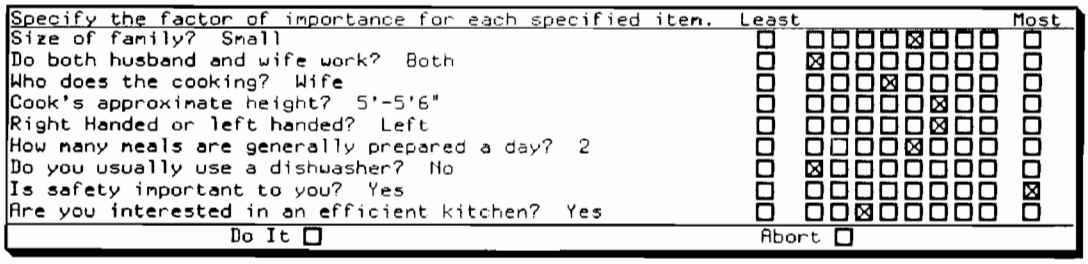
To use a partial specification to identify a relevant design object, one must consider the following issues: types of specification, weighting importance, and multiple contradictory features.

Types of specification

It has been observed that there exist two types of specification 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 of less than 100 ft²' or 'a kitchen that is good for a small child' are hidden features of the design that are not explicitly expressed in the final design artefact[23]. 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[30]. Hidden features can be computed or inferred only by the use of domain knowledge. There are two types of specifications in hidden features, *per se*. Features such as 'a



a



b

Figure 7. Specification component; (a) specification sheet, (b) weighting sheet for specification
[(a) The *specify* command in CatalogExplorer provides a specification sheet in the form of a questionnaire; (b) after specification, users weigh the importance of each specified item.]

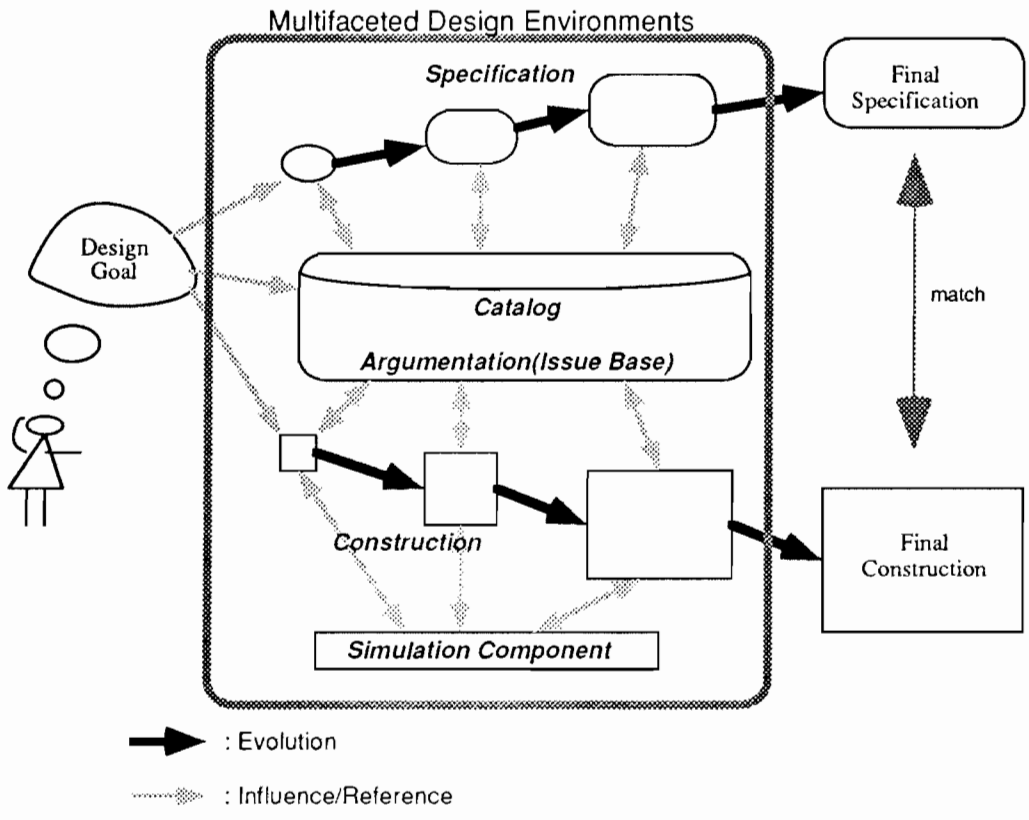


Figure 8. Coevolution of construction and specification of design in multifaceted architecture
[Starting with a vague design goal, designers go back and forth between the components in the environment. During the process, a designer and the system cooperatively evolve a specification and a construction incrementally by utilizing the available information in an argumentation component and a catalogue and feedback from a simulation component. In the end, the outcome is a matching pair of specification and construction. Sometimes, the 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.]

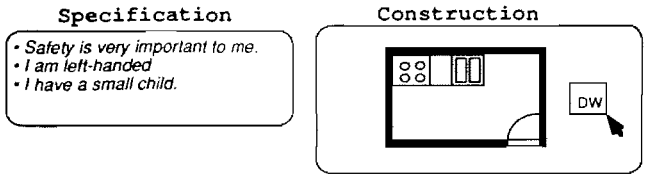


Figure 9. Task at hand in Janus

kitchen of less than 100 ft²' are objective or judgmental, whereas features such as 'a kitchen that is good for a small child' are subjective. A set of formal rules can be defined for the derivation of objective hidden features. In contrast, subjective hidden features can be inferred only relatively to one's viewpoint. That is, an inference of whether a kitchen design is good for a small child is subject to dispute, and may vary across time and society.

In practice, initial customer questionnaires given by professional kitchen designers to their customers often ask questions that relate to subjective hidden-feature specifications. 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, subjective hidden features are often represented in terms of a problem domain. Mechanisms for the retrieval of design objects from specifications should, therefore, be different, according to their type. Design examples can be retrieved from the catalogue by surface-feature specification with a conventional query mechanism, because they are already represented in a solution domain. In contrast, for retrieval of the design examples by hidden-feature specification, the system must have the domain knowledge to interpret these features into the solution structure.

Weighting importance

Sometimes, specified items contradict each other. If these contradictions are among hidden features, users may not notice the occurrence of the contradictions. Consequently, the system cannot retrieve design examples from the catalogue that satisfy their specification, because such examples do not exist. For example, consider the two specifications 'a safe kitchen' and 'a kitchen that provides easy access to the dining area'. Although these seem not to contradict each other, they do so in terms of hidden features. As seen in Figure 5, 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 the 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 users' specification than others.

Multiple contradictory features

One design object may have multiple contradictory features, i.e. hidden features that semantically contradict each other. For example, there can be a kitchen design in which some of the relationships of the 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 parts of a design may serve purposes that are contradictory to those of other parts of the same design.

Mechanisms

Specification-linking rules

CatalogExplorer automatically infers subjective hidden features of design examples in the catalogue 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 is away from door' and 'stove is away from window') that are conditions for a kitchen to have the 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 identify *a priori* which features would become relevant later. Such features may, however, become obsolete under new circumstances (e.g. an inefficient kitchen design may become efficient by the introduction of new appliances, such as a microwave cooker). Designers may not be able to articulate all the subjective features of a design, and, even if they could do so, such features may be difficult to understand.

The important aspect of the *specification-linking rules* is that these rules can be dynamically derived from the content of Janus Argumentation (see Figure 11). Suppose that the system has the following internal representation* for the *fire hazard* argument shown in Figure 5:

$$\neg\,(\text{Away-from-p STOVE DOOR}) \rightarrow \text{FIRE-HAZARDOUS} \quad (1)$$

and the system has the domain knowledge†

$$\text{SAFETY} \rightarrow \neg\,\text{FIRE-HAZARDOUS} \quad (2)$$

When users 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 needs by the following inference. First, Expression 1 is equivalent to the following:

$$\neg\,\text{FIRE-HAZARDOUS} \rightarrow (\text{Away-from-p STOVE DOOR}) \quad (3)$$

Therefore, by the use of Expressions 2 and 3,

$$\text{Expression 2} \wedge \text{Expression 3} \rightarrow (\text{SAFETY} \rightarrow (\text{Away-from-p STOVE DOOR})) \quad (4)$$

Appropriateness to set of specifications

To deal with some of the issues mentioned above, CatalogExplorer provides a mechanism for assigning a weight

*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 that computes a distance between two design units, and returns TRUE iff it exceeds a certain amount.

†This should read 'for a kitchen to be safe, it needs to be *not* fire-hazardous'.

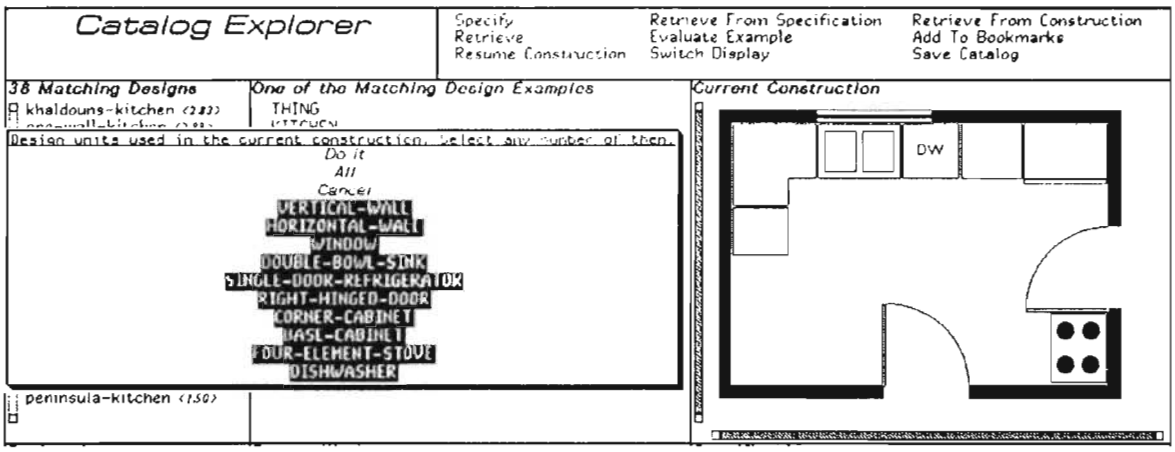


Figure 10. Retrieve from construction
[The *retrieve from construction* command, with a *parsing topic* 'design unit types', analyses the current construction, and provides a list of all the design unit types being used in the construction. Users can then select which design unit types they consider to be most important for the location of prestored designs in the catalogue.]

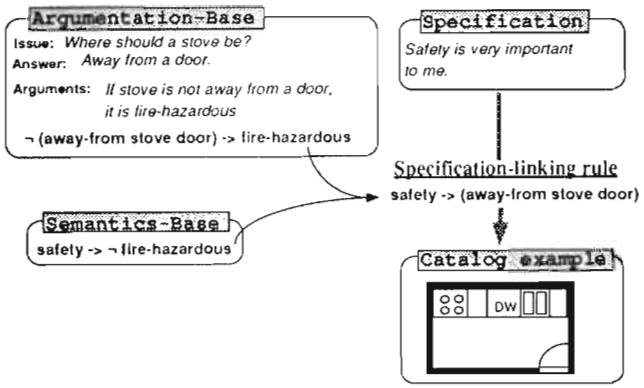


Figure 11. Specification-linking rules in CatalogExplorer

to each specification item, and it uses the concept of the *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 follows:

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

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

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

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

As a simple example, suppose that a user specifies one item 'is safety important to you? Yes' with a weight of 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 design that has a stove that is away from a door but close to a window is given the appropriateness value of $0.4=(1+0)/2\times 0.8$.

RETRIEVAL FROM CONSTRUCTION

For the retrieval of design examples that are related to a partial construction, one must deal with the issues of the matching of design examples in terms of the surface features of a design, i.e. at a structural level. The issues discussed in the previous section of the paper, 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. In other words, 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.

CatalogExplorer 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 first analyses the current construction, and then finds which design unit types are used, and provides the user with a menu to select some of them (see Figure 10).
- *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 that match this configuration element are retrieved.

RELATED WORK

The use of catalogues in design raises many problems in common with *case-based reasoning*. The authors' system serves as a *case-based decision-aiding system*[31] in a design domain rather than providing a mechanism for automated adaptation. The approach of case retrieval described in this paper offers some advantages over conven-

tional retrieval techniques by using synergy based on the integration.

Conventional retrieval techniques that are used in case-based reasoning systems are often applicable only to domains in which problems can be clearly articulated, such as word pronunciation[32]. These systems are unable to deal with fluctuations of problem specifications, and are inadequate for ill defined problems.

In Julia[33], problem and solution structures must be articulated in frame representations before a retrieval process is started. The *value frames* used in Julia provide the rationale behind a design decision. This can be used for the retrieval of cases. CatalogExplorer needs to integrate mechanisms to support the recording of the design rationale for this purpose[34].

Most case-based reasoning systems require representations of cases to be predetermined, and they are therefore not feasible. The approach presented in this paper addresses an indexing problem[23] by (a) focusing on *usefulness* rather than *structural similarities*, (b) combining abstract and surface features, and (c) providing dynamic indexing, which means putting the index at the retrieval time rather than the compilation time. The use of the specification-linking rules can be regarded as a type of analogical matching, such as the *systematicity-based match* in Cyclops[35]. In Cyclops, however, the explanations associated with cases must be predetermined, and cannot be dynamically computed.

The Interface system[36] provides interesting mechanisms for addressing some of the issues that relate 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 that of differentiating 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 the authors' system enable users to deal with more complex contradictory features. As it was built for the purpose of constructing a case-based library, the Interface system supported these mechanisms only while storing cases in the library. In the authors' work, the retrieval processes are driven by the user's task at hand, requiring that the weights be determined at the retrieval time, rather than at the time when the 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. Consequently, CatalogExplorer provides causal relationships between situations (specification) and solutions (constructions).

By the use of the environment over time, cases are collected incrementally. The system allows users to store design examples in the catalogue without checking for duplications and redundancies. Other systems store only prototypes[30], or prototypes and a small number of examples that are variations of them[36]. These approaches allow users to access *good* examples easily, and prevent the chaotic growth of the size of the catalogue. However, by not including failure cases, these catalogues prevent users from learning what went wrong in the past.

The integration relieves users of the task of specifying their goals for case retrieval. The task is articulated by other components in the environment.

DISCUSSION OF CATALOGEXPLORER

Achievements

In CatalogExplorer, users gradually narrow a catalogue space. The system can dynamically infer subjective hidden features, and provide users with an explanation for the inference mechanism. The system retrieves examples that are similar to the current construction, providing users with further directions in which to proceed with 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 that is relevant to the task at hand without requiring the users to form queries. Control of, and responsibility for, the retrieval of information is shared between the user and the system[4].

The authors' design environments empower both inexperienced and experienced designers. The system is useful for inexperienced designers, because it supports learning on demand[37]. It is useful for experienced designers, because it allows them to accumulate incrementally domain knowledge into the system. The authors' belief (which is based on interaction with numerous 'experts') is that even expert knowledge is never complete, because design situations are idiosyncratic and unique.

Limitations

A major limitation of the current system is the relatively small size of the catalogue (which comprises fewer than 100 examples). Many problems of managing large spaces effectively have not been dealt with. However, the authors are concerned about the scarce cognitive resource of humans, and not much concerned about computational resources. A lack of mechanisms for associating formal representations with arguments forces the manual derivation of the *specification-linking rules*. The definition of appropriateness is limited, and it needs a more sophisticated mechanism, such as the spreading of activation[38]. The parsers for the analysis of partial constructions should be extended to deal with more abstract levels, such as an emerging shape (e.g. an L shape or U shape) that currently requires to be specified by the user. A combinatorial use of the structural features for the detection of emerging features should be explored, such as the connectionism approach described by Newton and Coyne[39].

Future work

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

- *Level of assembly:* The use of Janus by kitchen designers has shown that the designers work not only with design units, but with higher-level abstractions, such as cooking centres and clean-up centres. These centres should be integrated into the palette, eliminating the clear distinction between the elements in the

palette and the catalogue. The catalogue should contain not only completed designs, but also important partial designs. These extensions will require further consideration of such issues as how to focus on a solution[23].

- *Support for other transition links:* A partial specification can be used to determine the set of relevant arguments in the argumentation component, enabling one to rearrange dynamically the argumentation space. A link between construction and specification can reduce the set of relevant units that is displayed in the palette.
- *Extensions of the architecture:* The authors' design environment for user-interface design[14,21] has been improved greatly in its effectiveness by the introduction of 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 the development of design environments, domain knowledge should be built into a *seed.* As users use the environment constantly, this seed should be extended. Sophisticated mechanisms for end-user modifiability[16] are crucial for this evolution of seeded environments.

BEYOND THE MACHO APPROACH OF ARTIFICIAL INTELLIGENCE

The authors' primary goal is to build cooperative problem-solving systems that empower humans, rather than to build expert systems that replace them[4,40]. They have pursued this approach not only because (a) automation approaches have failed in many domains (e.g. software design[41], the machine translation of natural language[42]), or (b) serious doubts have been articulated about 'in principle' limitations of expert systems[43,6], but also because they are convinced that *humans enjoy 'doing' and 'deciding'.* People often enjoy the process, and not just the final product; they want to take part in something. This is why they build model trains, plan their vacations, and design their own kitchens.

Automation can be a two-edged sword. At one extreme, it is a servant, relieving humans of the tedium of low-level operations, and freeing them for higher cognitive functions. Many people do not enjoy checking documents for spelling errors, and they welcome the automation that is provided by spelling checkers in word processors. At the other extreme, automation can reduce the status of humans to that of 'button pushers', and can strip their work of its meaning and satisfaction. The challenge is to automate tasks that people consider tedious or uninteresting; these may change as technology changes.

The authors' approach is to build *knowledge-based design environments*, but these are very different from expert systems. They aim to inform and support the judgment of designers, not to 'deskill' them by judging or designing for them. Designers that use these systems are free to ignore, turn off and alter the critiques given by the systems.

Building cooperative problem-solving systems allows the authors to exploit the relative strengths of the two participants to their advantage, i.e. humans are creative and can put tasks into larger contexts, whereas computers are good and dependable as depositories and managers of large amounts of information (such as building codes, safety rules, or the functional and aesthetic principles in Janus). The authors are interested in building 'human-centered' cooperative problem-solving systems (rather than 'computer-centered' ones), and it is for this reason that Schön's approach is valuable, because he has been involved in finding psychological explanations of human design activities.

Impact of Schön's work on authors' approach

As is evident throughout this paper, the authors' thinking and work has been influenced by Schön[7,8], as well as others (e.g. Ehn[44], Lave[45], Rittel[2], Simon[9], Suchman[43] and Winograd and Flores[6]). The major principle of Schön's work, and its influence on the authors' work, can be summarized as follows:

- *Design is a conversation with the materials of a design situation:* This principle is operationalized by the creation of domain-oriented design environments that support human problem-domain communications[13]. The 'materials' of the design situation are not low-level computer abstractions, but objects with which the domain worker is familiar. The domain orientation acknowledges that knowledge does not exist by itself in the form of context-free information, but is part of the practice of specific communities.
- *Situations need to talk back:* The 'back talk' is provided by the design situation itself, as well as by agents (who are, in Schön's case, humans, and in the authors' case, computational critics). It is important for the 'back talk' that it is relevant to the actual design situation, and that it is articulated in such a way that the designer can understand it.
- *Reflection in action:* The authors pay tribute to this concept by integrating construction and argumentation with the help of critics[46,47]. This integration is important, because (a) it creates a context-sensitive mechanism that provides entry into the hypermedia issue base in which argumentation that is relevant to the constructive design situation can be found, and (b) it makes designers aware that they may need additional information. Reflective processes are triggered by violations of the principles of design, but the authors' systems allow reflection on the principles of design themselves, as well.
- *Integration of problem setting and problem solving:* Schön shares the belief with other design methodologists, such as Rittel[2], (and provides empirical evidence for it) that practitioners who solve problems do not operate in a given solution space, but that this space is incrementally constructed in response to solution attempts.
- *Design knowledge is tacit:* Competent practitioners usually know more than they can say. Their tacit knowledge is triggered by new design situations, and by breakdowns that occur as they engage in a design process. This requires that design worlds be not closed, but open-ended. This leads to developments in the authors' work to support end-user modifiability[16] and the evolution of design environments (starting with seeded environments) in response to new design problems.

- *Emphasis on the different demands of rigour versus relevance:* Schön illustrates the different demands of rigour versus relevance, and shows that practitioners need to be more concerned with relevance than rigour. Although the authors' critiquing systems and case libraries are weak with respect to formal rigorous criteria (such as completeness and consistency), they provide relevant information in actual design situations[48].

Moving beyond Schön's work

The authors' work starts with the principles mentioned in the previous sections of the paper, and asks 'how can they be facilitated in computer-based tools?'. Schön's interest is not in building systems that assist designers in design tasks; he is interested in finding psychological explanations of the designer. His theory is descriptive, and it identifies the importance of human resources in this process (e.g. as illustrated by the scenario between Petra and Quist, in which Quist acts as a critic for Petra (Reference 7, pp 79–104)).

The authors' interest is in understanding how designers design, how designers might organize designing so that they are more effective, avoid problems, and learn new things as they go along, and how all of this can be supported by computational media. They have engaged in something that Schön's theory considers important: building objects to think with in the forms of demonstration prototypes (design environments) to test the theory in practice, experience breakdowns of the theory, and, as a consequence, refine the theory when necessary. Schön's own work has not followed his theory: the intertwining of theory building (reflection) with theory instantiation (action) would make it so. In the authors' work, they have demonstrated that computational mechanisms can be created that can take some of Schön's concepts and bring them alive in a computational environment.

Importance of domain-oriented architecture

Skilled domain workers, unlike designers, know what the job is. However, they do not know what can be designed. If this observation is correct, then 'design should be done with users, neither for them nor by them'. Work-oriented design[44] is based on the assumption that design for an 'ideal situation' is impossible; it is essential to design for the work that people do, rather than for a disembodied, idealized description of the work process. The authors support Schön's concept of knowing in action through the use of domain-oriented construction kits, which allow designers to work directly with the concepts of the problem situation itself, rather than requiring them to work with computer-oriented concepts. This approach pushes the computer into the background, and turns it into an invisible instrument, which Schön describes as necessary for knowing in action.

Critiquing systems

Schön's framework is based on the basic cycle of 'seeing–drawing–seeing'. However, Schön's notion of *seeing* is 'not good enough'; as Rittel pointed out, 'buildings do not speak for themselves'. Nonexpert designers (and this is what designers are, in almost all realistic situations) do not have the complete knowledge and experience to understand fully the conversation with the materials of the situation. Critiquing mechanisms[49] serve as 'interpreters' that support designers in seeing and understanding the 'back talk' of the situation. When a critic fires, reflection does not occur on the simple basis of the message. Designers 'listen to' the design material with the help of the interpreter in the form of a critic. The authors' critiquing systems address the problem of the situations created by technological advances and the division of labour — situations in which more and more designers deal with objects that do not primarily belong to their communities of practice.

Critics provide a computational mechanism that allows designers to think about what they are doing while this thinking can still make a difference. For a situation to talk back, a human has to understand the information that is being given. The difference between 'feedback of the system' and 'back-talking situations' is related to the understanding of humans. Relevancy to the task at hand seems to play an important role here, because the more given information is relevant to the current problem situation, the more understandable the information is for a human.

Integrated design environments

The authors' design environments can exploit the information that is contained in partial specifications and partial constructions to increase the contextualization of the materials of the situation to the task at hand. Work objects, which are explicitly represented in the authors' environments, play a crucial role in cooperation between designers, designers and clients, and designers and users. The lack of the work objects in nonintegrated, detached reflection-support systems (e.g. gIBIS[50]) makes it impossible for users to access relevant information in terms of the artefact.

Schön describes knowing in action as a spontaneous, nonreflective and unselfconscious engagement in an action such as the construction of a solution form. According to him, it cannot take place when designers are forced to reflect on every move that they make, or when the tools become too obtrusive. The authors support knowing in action by separating the construction and argumentation components so that designers can construct the solution form without explicit reflection, unless there is a breakdown of knowing in action.

Evolution

Design knowledge is tacit, and design worlds are open-ended. Users of design environments must be able to extend them in response to breakdowns. The human practice of carrying out tasks evolves over time, and a cooperative problem-solving system must be adaptable and/or adaptive to reflect these changes. The boundary between what a human does and what the system does in a semiformal architecture changes over time, because the understanding of what can be formalized grows over time as people reflect on their jobs and establish routines. The authors are in the process of constructing seeded environments for domains that serve as the background information against which specific design projects can be carried out. At the same time, each new design project contributes toward the further development of the seed.

Exploiting unique possibilities of computers as medium
In noncomputational environments, 'seeing' can be enhanced by training, or supported by a human. Computational environments create the unique opportunity to place some of the subjective seeing burden on the computation. The environments need to remediate for the perceptually untrained, and engage them in reflective conversation at the level that they can handle, and teach them to see. Beyond remediation for lack of perceptual training, there are many facts about designs that even the most well trained can never see directly, but that computation can visualize for them. If designers are willing to annotate their work products, computers can deliver this additional information to future designers[47].

Issues for further investigation

By engaging in reflection in action with the use of computational environments, the authors have created situations that talk back to them. Their system-building efforts, and the use of those systems, create breakdowns, which trigger further reflection, creating a large number of interesting issues that should be pursued. Among these are careful studies of the use of the authors' environments by domain workers that should investigate the following specific issues:

- Are there differences in the performance and quality of the product if the system is used with and without critics, the catalogue and the simulation components?
- What are the tradeoffs between running the system in a critiquing mode or a constraint mode[51], where the latter prevents certain problems from arising (e.g. by enforcing building codes), whereas the former provides designers with opportunities of dealing with breakdowns?
- What are the tradeoffs between different intervention strategies, e.g. the balance between displaying enough information versus the disruption of the work process? When are designers willing to suspend the construction process to access relevant information? Does 'making information relevant to the task at hand' prevent serendipity?
- If an environment can always supply the information that the situation demands, why will users bother to learn the information[37]? How can working and learning be better integrated[52]?
- Under which conditions will designers challenge or extend the knowledge represented in the system? How can they be motivated to do so?
- Should the 'back talk' be embedded directly into the artefact, or handled by a separate discourse? It is conceivable that diving into hypermedia focuses users on other tasks, and takes them out of the situation.
- If information is plentiful, what is scarce? How can information-delivery systems be created that make information more relevant to the task at hand?
- 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[44,43]? People do use plans (e.g. milestone charts and business plans). Designers, by making and following systematic agreements (rules) about the selection and position of elements, can work more effectively as a team. Even if one agrees that 'design is more than the application of standard principles', one cannot infer that principles cannot be useful.

CONCLUSIONS

Design activities incorporate many cognitive issues, such as the recognition and framing of a problem, the understanding of given information, and the adaptation of the information to the situation. The integration of problem setting and problem solving is crucial in dealing with ill defined problems. This paper has described mechanisms that relate partial specifications and partial constructions to a catalogue of prestored designs, thereby retrieving design objects stored in a catalogue that are relevant to the task at hand without the users being asked to form queries. The system demonstrates the synergy of integrated design environments, empowering human designers. It does not force human designers to use the mechanisms described above. The authors' environments provide mechanisms and resources that are available whenever humans need them. As a number of research efforts have demonstrated, the multifaceted architecture developed in the context of this work is a promising architecture for the building of a great variety of integrated design environments in different domains.

The work has profited from the conceptual framework of Donald Schön, and, at the same time, it has tried to extend his ideas by instantiating them with computational mechanisms. Design should be understood and practised as a 'dialectical process between tradition and transcendence'[44]. From this perspective, Schön's work is limited, because it neither suggests, nor tries to invent, design methodologies and cooperative problem-solving systems that are driven by how things might be done differently, rather than by how things have always been done.

ACKNOWLEDGEMENTS

The authors would like to thank Mark Gross and Raymond McCall (University of Colorado at Boulder, USA), Will Hill (Bellcore), Anders Morch (Nynex), Loren Terveen (AT&T, USA) and Dave Wroblewski (USWest, USA) for their insightful comments on the relationship of Schön's work to the authors' work. The authors would also 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 US National Science Foundation under Grants IRI-8722792 and IRI-9015441, and by the US Army Research Institute under Grant MDA903-86-C0143.

REFERENCES

1 **Simon, H A** 'The structure of ill-structured problems' *Artif. Intell.* No 4 (1973) pp 181–200
2 **Rittel, H W J** 'Second-generation design methods' *in* Reference 3, pp 317–327
3 **Cross, N** *Developments in Design Methodology* John Wiley, USA (1984)
4 **Fischer, G** 'Communications requirements for coo-

perative problem solving systems' *Int. J. Inf. Syst.* Vol 15 No 1 (1990) pp 21–36 (special issue on knowledge engineering)
5 **Stefik, M J** 'The next knowledge medium' *AI Magazine* Vol 7 No 1 (1986) pp 34–46
6 **Winograd, T and Flores, F** *Understanding Computers and Cognition: A New Foundation for Design* Ablex, USA (1986)
7 **Schön, D A** *The Reflective Practitioner: How Professionals Think in Action* Basic Books, USA (1983)
8 **Schön, D A** *Educating the Reflective Practitioner* Jossey–Bass, USA (1987)
9 **Simon, H A** *The Sciences of the Artificial* MIT Press, USA (1981)
10 **Fischer, G and Reeves, B N** 'Beyond intelligent interfaces: exploring, analyzing and creating success models of cooperative problem solving' *Appl. Intell.* (in press) (special issue on intelligent interfaces)
11 **Sheil, B A** 'Power tools for programmers' *Datamation* (Feb 1983) pp 131–144
12 **Hutchins, E L, Hollan, J D and Norman, D A** 'Direct manipulation interfaces' *in* **Norman, D A and Draper, S W (Eds.)** *User Centered System Design, New Perspectives on Human-Computer Interaction* Lawrence Erlbaum, USA (1986) pp 87–124
13 **Fischer, G and Lemke, A C** 'Construction kits and design environments: steps toward human problem-domain communication' *Human-Comput. Interact.* Vol 3 No 3 (1988) pp 179–222
14 **Lemke, A C** 'Design environments for high-functionality computer systems' *PhD Dissertation* Dep. Computer Science, University of Colorado at Boulder, USA (Jul 1989)
15 **Nielsen, J and Richards, J T** 'The experience of learning and using Smalltalk' *IEEE Software* (May 1989) pp 73–77
16 **Fischer, G and Girgensohn, A** 'End-user modifiability in design environments' *Proc. CHI'90 Conf. Human Factors in Computing Systems* ACM, USA (1990) pp 183–191
17 **Fischer, G, Henninger, S and Redmiles, D** 'Intertwining query construction and relevance evaluation' *Proc. CHI'91 Conf. Human Factors in Computing Systems* ACM, USA (1991) pp 55–62
18 **Henninger, S** 'Defining the roles of humans and computers in cooperative problem solving systems for information retrieval' *Proc. AAAI Spring Symp. Wkshp. Knowledge-Based Human-Computer Communication* (Mar 1990) pp 46–51
19 **Halasz, F G** 'Reflections on NoteCards: seven issues for the next generation of hypermedia systems' *Commun. ACM* Vol 31 No 7 (1988) pp 836–852
20 **Fischer, G, McCall, R and Morch, A** 'JANUS: integrating hypertext with a knowledge-based design environment' *Proc. Hypertext'89* ACM, USA (1989) pp 105–117
21 **Lemke, A C and Fischer, G** 'A cooperative problem solving system for user interface design' *Proc. 8th Nat. AAAI'90 Conf. Artificial Intelligence* AAAI Press, USA (1990) pp 479–484
22 **McCall, R** 'Issue-serve systems: a descriptive theory of design' *Des. Methods & Theor.* Vol 20 No 8 (1986) pp 443–458
23 **Kolodner, J L** 'What is case-based reasoning?' *8th Nat. AAAI'90 Conf. Artificial Intelligence Tutorial Text on Case-Based Reasoning* AAAI Press, USA (1990) pp 1–32
24 **Riesback, C K and Schank, R C** *Inside Case-Based Reasoning* Lawrence Erlbaum, USA (1989)
25 **Fischer, G, Lemke, A C, Mastaglio, T and Morch, A** 'Using critics to empower users' *Proc. CHI'90 Conf. Human Factors in Computing Systems* ACM, USA (1990) pp 337–347
26 **Rissland, E L and Shalak, D B** 'Combining case-based and rule-based reasoning: a heuristic approach' *Proc. 11th Int. Joint Conf. Artificial Intelligence* Morgan Kaufmann, USA (1989) pp 524–530
27 **Patel-Schneider, P F** 'Small can be beautiful in knowledge representation' *AI Technical Report 37* Schlumberger Palo Alto Research, USA (Oct 1984)
28 **Fischer, G and Nieper-Lemke, H** 'HELGON: extending the retrieval by reformulation paradigm' *Proc. CHI'89 Conf. Human Factors in Computing Systems* ACM, USA (1989) pp 357–362
29 **Williams, M D** 'What makes RABBIT run?' *Int. J. Man-Machine Stud.* Vol 21 (1984) pp 333–352
30 **Gero, J S** 'Design prototypes: a knowledge representation schema for design' *AI Magazine* Vol 11 No 4 (1990) pp 26–36
31 **Kolodner, J L** 'Improving human decision making through case-based decision aiding' *AI Magazine* Vol 12 No 2 (1991) pp 52–68
32 **Stanfill, C and Waltz, D L** 'The memory-based reasoning paradigm' *Proc. Case-Based Reasoning Wkshp.* Morgan Kaufmann, USA (1988) pp 414–424
33 **Kolodner, J L** 'Extending problem solving capabilities through case-based inference' *Proc. Case-Based Reasoning Wkshp.* Morgan Kaufmann, USA (1988) pp 21–30
34 **Fischer, G, Lemke, A C, McCall, R and Morch, A** 'Making argumentation serve design' *Technical Report* Dep. Computer Science, University of Colorado at Boulder, USA (1991)
35 **Navinchandra, D** 'Case-based reasoning in CYCLOPS' *Proc. Case-Based Reasoning Wkshp.* Morgan Kaufmann, USA (1988) pp 286–301
36 **Riesbeck, C K** 'An interface for case-based knowledge acquisition' *Proc. Case-Based Reasoning Wkshp.* Morgan Kaufmann, USA (1988) pp 312–326
37 **Fischer, G** 'Supporting learning on demand with design environments' *Proc. Int. Conf. Learning Sciences 1991* Evanston, IL, USA (Aug 1991) pp 165–172
38 **Henninger, S** 'Retrieving software objects in an example-based programming environment' *Proc. SIGIR'91* Chicago, IL, USA (October 1991) (in press)
39 **Newton, S and Coyne, R D** 'Impact of connectionist systems on design' *in* **Gero, J (Ed.)** *Artificial Intelligence in Design* Butterworth–Heinemann, UK (1991) pp 49–75
40 **Fischer, G and Nakakoji, K** 'Making design objects relevant to the task at hand' *Proc. 9th Nat. AAAI'91 Conf. Artificial Intelligence* AAAI Press, USA (1991) pp 67–73
41 **Barstow, D** 'A perspective on automatic programming' *Proc. 8th Int. Joint Conf. Artificial Intelligence* Morgan Kaufmann, USA (1983) pp 1170–1179

42 **Kay, M** 'The proper place of men and machines in language translation' *Technical Report CSL-80-11* Xerox Palo Alto Research Center, USA (Oct 1980)
43 **Suchman, L A** *Plans and Situated Actions* Cambridge University Press, UK (1987)
44 **Ehn, P** *Work-Oriented Design of Computer Artifacts* Almquist & Wicksell, Sweden (1988)
45 **Lave, J** *Cognition in Practice* Cambridge University Press, UK (1988)
46 **McCall, R, Fischer, G and Morch, A** 'Supporting reflection-in-action in the Janus Design Environment' *in* **McCullough, M *et al.* (Eds.)** *The Electronic Design Studio* MIT Press, USA (1990) pp 247–259
47 **Fischer, G, Grudin, J, Lemke, A C, McCall, R, Ostwald, J and Shipman, F** 'Supporting asynchronous collaborative design with integrated knowledge-based design environments' *Technical Report* Dep. Computer Science, University of Colorado at Boulder, USA (1991)
48 **Simon, H A** 'Alternative representations for cognition: search and reasoning' *Technical Report* Dep. Psychology, Carnegie Mellon University, USA (1991)
49 **Fischer, G, Lemke, A C, Mastaglio, T and Morch, A** 'The role of critiquing in cooperative problem solving' *ACM Trans. Inf. Syst.* (in press)
50 **Conklin, J and Begeman, M** 'gIBIS: a hypertext tool for exploratory policy discussion' *Proc. Conf. Computer Supported Cooperative Work* ACM, USA (1988) pp 140–152
51 **Gross, M D and Boyd, C** 'Constraints and knowledge acquisition in Janus' *Technical Report* Dep. Computer Science, University of Colorado at Boulder, USA (1991)
52 **Fischer, G, Lemke, A C and McCall, R** 'Towards a system architecture supporting contextualized learning' *Proc. 8th Nat. AAAI'90 Conf. Artificial Intelligence* AAAI Press, USA (1990) pp 420–425