Making Design Objects Relevant to the Task at Hand

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

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

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

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

Abstract

Many problem-solving approaches are based on the assumption that a problem can be precisely defined before it is solved. These approaches are inadequate for dealing with ill-defined problems, which require the coevolution of problem setting and problem solving. In this paper, we describe integrated, domain-oriented, knowledge-based design environments and their underlying *multifaceted architecture*. The environments empower humans to cope with ill-defined problems, such as design, by supporting an incremental approach to problem setting and problem solving. We focus on the integration of specification, construction, and a catalog of prestored design objects in those environments. The synergy of integration enables the environments to make those objects relevant to the task at hand. Taking architectural design as a domain to illustrate our approach, we describe an operational, prototype system (CATALOGEXPLORER) that assists designers in locating examples in the catalog that are relevant to the task at hand as articulated by a partial specification and a partial construction. Users are thereby relieved of the task of forming queries and navigating in information spaces.[1]

Introduction

Artificial intelligence has often been characterized as the discipline dealing with ill-defined problems (Simon, 1973). Problem-solving approaches that are based on directionality, causality, and separation of analysis from synthesis are inadequate for solving ill-defined problems (Cross, 1984). *Design* is an example of such an ill-defined problem. Our work is based on the premise that design problems are best solved by supporting a cooperative problem-solving approach between humans and integrated, domain-oriented, knowledge-based design environments (Fischer, 1990). Combining knowledge-based systems and innovative human-computer communication techniques empowers humans to produce "better" products by augmenting their intellectual capabilities and productivity rather than by replacing them with an automated system (Stefik, 1986; Winograd & Flores, 1986).

In this paper, we will use the domain of architectural design of kitchen floor plans as an "object-to-think-with" for purposes of illustration. The simplicity of the domain allows us to concentrate on the issues of our approach without being distracted by understanding the domain itself. We discuss ill-defined problems, emphasize the importance for design environments to be domain-oriented and integrated, and describe the multifaceted architecture. We describe CATALOGEXPLORER, which integrates specification, construction, and a catalog of prestored design objects. The system demonstrates the synergy of the integration by showing that it can partially articulate the user's task at hand by a partial specification and focus the user's attention on design objects that are relevant to that task. We conclude with a discussion of our own and other related work, and future directions.

Coping with Ill-defined Problems

Ill-defined Problems

Most design problems are ill-defined (Hayes, 1978; Cross, 1984; Rittel & Webber, 1984; Swartout & Balzer, 1982). Such problems require the coevolution of problem setting and problem solving. The information needed to understand the problem depends on one's idea for solving it, and vice versa. Professional practitioners spend at least as much time in defining the problem as in solving the problem (Schoen, 1983; Rittel, 1984). Every step made toward a solution creates a new problem, providing humans with a continuing source of new ideas (Simon, 1981). Expert systems and automated problem-solving technologies fail in coping with ill-defined problems because they need to identify the information required for a solution a priori.

Cooperative Problem Solving

An empirical study of our research group, which analyzed *human-human cooperative problem solving* in a large hardware store (Reeves, 1990), provided ample evidence that in many cases humans are initially unable to articulate

[1]The research was supported by Software Research Associates, Inc. (Tokyo, Japan), by the National Science Foundation under grants No. IRI-8722792 and IRI-9015441, and by the Army Research Institute under grant No. MDA903-86-C0143.

complete requirements. They start from a partial specification and refine it incrementally, based on the feedback they get from their environment. Because users are actively involved in problem setting and problem solving processes, there is a necessity for systems to support the task at a level that is comprehensible by the user.

Domain Orientation

To reduce the transformation distance between a design substrate and an application domain (Norman, 1986), designers should be able to perceive design as communication with an application domain. The computer should become invisible by supporting *human problem-domain communication*, not just human-computer communication (Fischer & 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 computing environment. In such an environment, designers build artifacts from application-oriented building blocks according to the principles of the domain, reducing the large transformation distance between problem formulation and computational environment.

Information Relevant to the Task at Hand

In supporting integration of problem setting and problem solving in design environments, supporting retrieval of information relevant to the task at hand is crucial. Every step made by a designer toward a solution determines a new space of related information, which cannot be determined a priori due to its very nature.

Conventional information retrieval techniques are thus not applicable for design environments (Fischer, Henninger, & Redmiles, 1991). In a conventional *query-based search*, a specific query has to be formulated. Once users can articulate what they need, a query-based search takes away much of the burden of locating promising objects (Henninger, 1990). In *navigational access* provided by a browsing mechanism, users tend to get lost looking for some target information if the browsing space is large and the structure is complex (Halasz, 1988). Navigational access requires that the information space 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 identifying a small set of relevant information.

Design environments, therefore, need additional mechanisms (as discussed in this paper) that can identify small sets of objects relevant to the task at hand. The systems must be able to support users to incrementally articulate the task at hand. The information provided in response to these problem-solving activities must assist users in refining the definition of their problem.

Integrated, Domain-Oriented, Knowledge-Based Design Environments

A Multifaceted Architecture

During the last five years, we have developed and evaluated several prototypes of domain-oriented design environments, for example, for architectural design (Fischer, McCall, & Morch, 1989) and for user interface design (Lemke & Fischer, 1990). The different system-building efforts led to the multifaceted architecture shown in Figure 1.

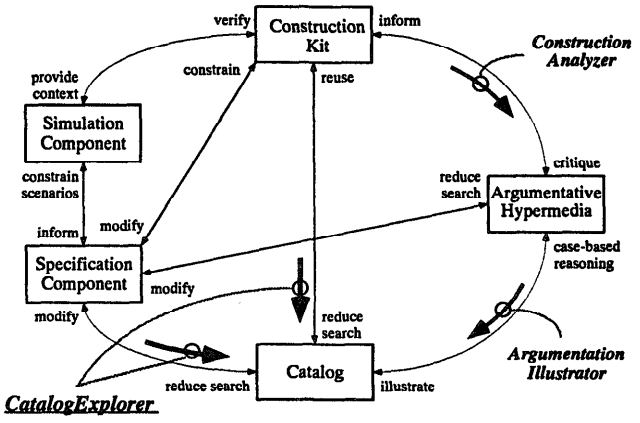

Figure 1: A Multifaceted Architecture

This figure shows the components of the multifaceted architecture. The links between the components are crucial for exploiting the synergy of the integration.

The multifaceted architecture consists of the following five components (for details see Fischer, McCall, and Morch (1989), and Fischer (1990)):

- A *construction kit* provides a palette of domain abstractions and supports the construction of artifacts using direct manipulation and other interaction styles. This is the principal medium for implementing design reflecting a user's current problem situation. Completed designs can be stored in the catalog for reuse.
- An *argumentative hypermedia system* contains issues, answers, and arguments about the design domain.
- A *catalog* provides a collection of prestored design objects illustrating the space of possible designs in the domain. Catalog examples support reuse and case-based reasoning (Riesbeck & Schank, 1989; Slade, 1991).
- A *specification component* allows designers to describe some characteristics of the design they have in mind. The specifications are expected to be modified and augmented during the design process, rather than to be fully articulated at the beginning. They are used to prioritize all other information spaces in the system with respect to the emerging task at hand.
- A *simulation component* allows users to carry out

"what-if games" to simulate usage scenarios with the artifact being designed. Simulation complements the argumentative component, which cannot capture all relevant aspects in the situation.

Integration

The multifaceted architecture derives its essential value from the integration of its components and links between the components. Each component augments the value of the others, forming a synergistic whole. At each stage in the design process, the partial design embedded in the design environment serves as a stimulus to users for suggesting what they should attend to next. The multifaceted architecture supports that "*situations talk back*" to users by providing them with immediate and task-relevant feedback (Schoen, 1983).

Links among the components of the architecture are supported by various mechanisms (see Figure 1). A user's task at hand can be partially articulated in each component of the environment. Consequently the integration enables the system to provide the user with the information relevant to the task at hand. The major mechanisms to achieve this are:

- *CONSTRUCTION ANALYZER* consists of a set of critics (Fischer et al., 1990) that detect and critique partial solutions constructed by the users. The firing of a critic signals a breakdown to users and provides them with immediate entry into the exact place in the argumentative hypermedia system at which the corresponding argumentation is located.
- *ARGUMENTATION ILLUSTRATOR* helps users to understand the information given in an argumentative hypermedia by using a catalog design example as a source of concrete realization (Fischer, 1990). 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* helps users to search the catalog space according to the task at hand. It retrieves design examples similar to the current construction situation, and orders a set of examples by their appropriateness to the current specification.

A typical cycle of events supported by the multifaceted architecture is: (1) users create and refine a partial specification or construction, (2) a breakdown occurs, (3) users switch and consult other components in the system made relevant by the system to the partially articulated task at hand, and (4) users refine their understanding based on "back talk of the situation" (Schoen, 1983). As users go back and forth among these components, the problem space is narrowed, a shared understanding between users and the system evolves, and the artifact is incrementally refined.

CATALOGEXPLORER

CATALOGEXPLORER links the specification and construction components with the catalog in JANUS (see Figure 1). CATALOGEXPLORER (1) exploits the information articulated in a partial specification to prioritize the designs stored in the catalog, and (2) analyzes the current construction and retrieves *similar* examples from the catalog using similarity metrics. Figure 2 shows a screen image of the system. Each design object stored in the catalog of JANUS consists of a floor layout and a set of slot values filled by users. Those design objects can be reused for case-based reasoning such as providing a solution to a new problem, evaluating and justifying decisions behind the partial specification or construction, and informing designers of possible failures (Slade, 1991; Kolodner, 1990).

Based on the HELGON system (Fischer & Nieper-Lemke, 1989), CATALOGEXPLORER stores the design examples as objects in a KANDOR knowledge base (Patel-Schneider, 1984). Examples are automatically classified according to their features specified as slot values. The system supports *retrieval by reformulation* (Williams, 1984), which allows users to incrementally improve a query by critiquing the results of previous queries.

CATALOGEXPLORER extends HELGON and other existing information retrieval systems by relieving users of the task of forming queries and navigating in information spaces. Being integrated based on the multifaceted architecture, CATALOGEXPLORER can capture a user's task at hand by analyzing the partial specification and construction. The system, then, computes and infers the relevance of stored information to that task. In general, the relevance cannot be determined objectively in dealing with ill-defined problems because we cannot completely identify such relevant factors. What has been made explicit always sets a limit, and there exists the potential for breakdowns that call for moving beyond this limit (Winograd & Flores, 1986). For overcoming issues of using a fixed set of rules for inferring the relevance, therefore, one of our current efforts is focusing on dynamically deriving such rules by the domain knowledge stored by users in the argumentation component.

In the rest of this section, we describe the mechanism of making design objects relevant to the task at hand by using a partial specification. The retrieval mechanism using a partial construction is discussed in Fischer and Nakakoji (1991).

Retrieval from Specification. As a specification component in the multifaceted architecture, CATALOGEXPLORER provides (1) a *Specification Sheet* for specifying requirements for a design (see Figure 3), and (2) a *Weighting Sheet* for assigning a weight to each specification item to differentiate the factor of importance (see Figure 4). By analyzing given information by those mechanisms, the system reorders catalog examples by computing the *appropriateness* value of each design example according to the given set of weighted specifications (see the *Matching Designs* window in Figure 2).

For capturing the user's task at hand from a specification

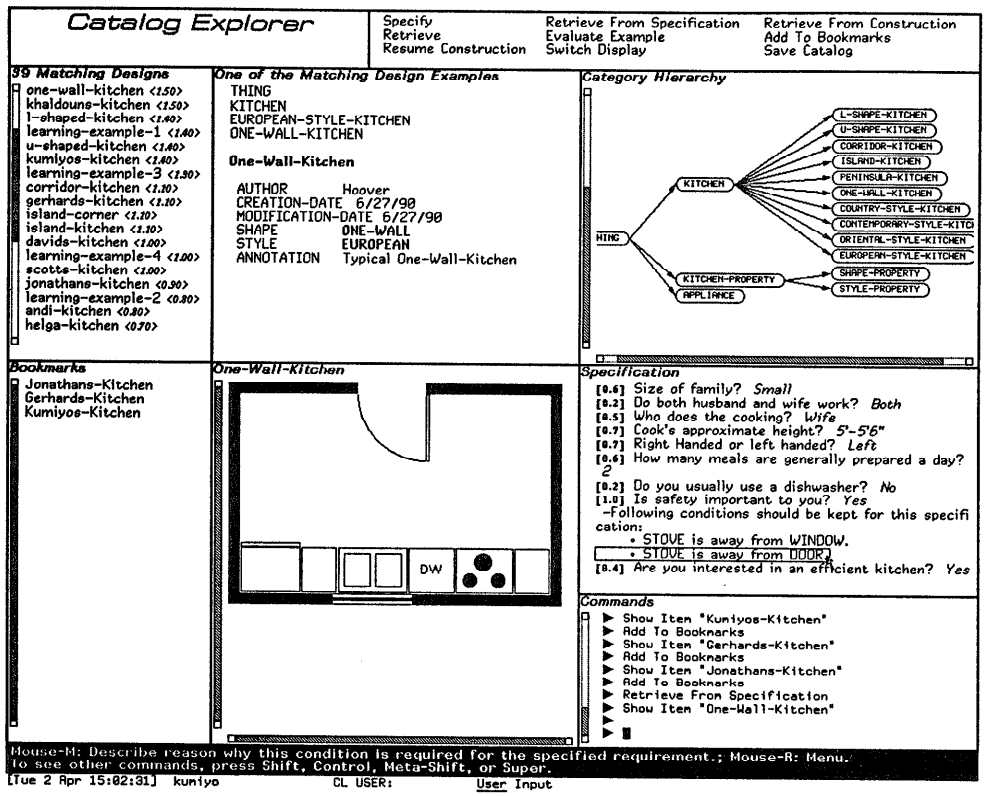

Figure 2: A Screen Image of CATALOGEXPLORER

The leftmost *Matching Designs* window lists all currently retrieved design examples in the catalog, ordered according to appropriateness to the current specification. The *Bookmarks* window is used as a temporary name holder of catalog 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 catalog. The *Specification* window shows specified items with the assigned weight of importance (result of Figures 3 and 4). Items in this window are mouse-sensitive, and by clicking on one, CATALOGEXPLORER provides the information of the corresponding *specification-linking rules* (two lines in the middle of the window). Clicking on one of the rules will activate JANUS-ARGUMENTATION providing the underlying argumentation for that rule (see Figure 5).

and making design objects relevant to that task by inferring the relevance, one must deal with hidden features, partial matching, and contradictory features of design. To address these issues, the system has *specification-linking rules* for matching between a specification and design objects, and a metric to measure the *appropriateness* of an existing design with respect to a specification.

Specification-linking Rules. CATALOGEXPLORER supports users in retrieving catalog examples by hidden feature specifications (see Figure 3) by using *specification-linking rules.*

There are two types of specification items: *surface features* such as *"a kitchen that has a dishwasher"* and *hidden features* such as *"good for a small family."* Retrieving design examples from the catalog by surface feature specification can be done in a straightforward manner using conventional searching mechanisms. In contrast, retrieval using hidden features requires domain knowledge to infer those features because it is often difficult to determine a priori the features that become important for later recall. Hidden features can be classified into *objective* and *subjective* ones; the former ones can be derived by a set of predefined formal rules, whereas the latter need to be dynamically inferred because they are subject to dispute and may vary across time and society.

The *specification-linking rules* of CATALOGEXPLORER link each subjective hidden feature specification item to a set of physical condition rules. In Figure 2, in the *Specification* window, the shown rules indicate that a kitchen that has a stove away from both a door and a window satisfies a hidden feature such as *a safe kitchen.*

In the integrated environment this domain knowledge can be derived from the content of the argumentative hypermedia component. Suppose users provided the system with the following formal representation to the *"Fire Hazard"* argument in Figure 5,

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

and the system has the domain knowledge such as:

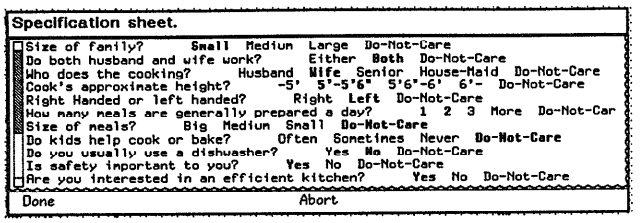

Figure 3: Specification Sheet

The *Specify* command in CATALOGEXPLORER provides a specification sheet in the form of a questionnaire.

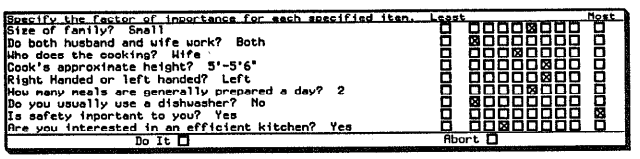

Figure 4: Weighting Sheet for the Specification

After specification, users have to weigh the importance of each specified item.

SAFETY → ¬ FIRE-HAZARDOUS (2)

Then, when users specify that they are concerned about safety, the system infers that design examples with a stove away from a door are appropriate as follows:

(1) ■ (¬ FIRE-HAZARDOUS → (Away-from-p STOVE DOOR)) (3)

(2) ∧ (3) → (SAFETY → (Away-from-p STOVE DOOR)) (4)

The specified items (see the *Specification* window in Figure 2) are associated with a set of specification-linking rules, and each of those rules is associated with corresponding arguments in JANUS-ARGUMENTATION. Thus, users can freely explore the underlying inference mechanisms by simply clicking on a displayed rule that provides users with an exact entry in JANUS-ARGUMENTATION.

Those specification-linking rules could be dynamically augmented by being derived from arguments in the argumentative hypermedia component. Suppose that users come up with a new argument in JANUS-ARGUMENTATION and provide the system with a corresponding formal representation as follows:

Issue: *Where should the location of a stove be?*
Answer: *Not next to a refrigerator.*
Argument: *A stove next to a refrigerator is fire hazardous because one's clothes may accidentally catch fire from a stove while looking into the refrigerator.*

(Next-to-p STOVE REFRIGERATOR) → FIRE-HAZARDOUS (5)

Then, the system adds a new condition to the specification rule for being *a safe kitchen*, and infers kitchens with a stove next to a refrigerator are *not* appropriate to the user's specification concerning safety.

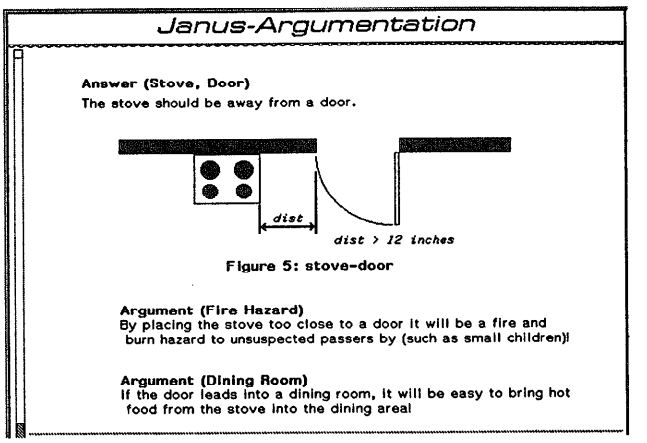

Figure 5: Corresponding Arguments in JANUS-ARGUMENTATION

JANUS-ARGUMENTATION is the argumentative hypermedia component of JANUS.

Appropriateness to a Set of Specifications. To deal with partial matching and contradictory features of a design object, CATALOGEXPLORER provides a mechanism for assigning a weight to each specification item and uses the concept of *appropriateness* of a design example to a set of specification items. The appropriateness of a design in terms of a set of specification items is defined as the weighted sum of the number of satisfied conditions out of applicable specification-linking rules to each design (for details see Fischer and Nakakoji (1991)). By seeing the effects of changing the factor of importance in the ordered catalog examples, users can make tradeoffs among contradictory specification items.

Discussion of Our Approach

Related Work

Using catalogs in design raises many problems in *case-based reasoning* (Riesbeck & Schank, 1989; Slade, 1991). Retrieval techniques used in case-based reasoning systems, however, are often applicable only for domains in which problems can be clearly articulated, such as word pronunciation (Stanfill & Waltz, 1988), or in which problem and solution structures can be articulated in frame representations before starting a retrieval process (Kolodner, 1988).

Most existing case-based reasoning systems require representations of cases to be predetermined, and therefore are not feasible for design domains. Our work addresses the indexing problem by using more than a surface representation of a case, and the matching process operates at an abstract level of representation. The specification-linking rules support analogical matching (similar to a *systematicity-based match* (Navinchandra, 1988)). In our work, the explanations associated with cases can be

dynamically computed and do not need to be predetermined.

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

CATALOGEXPLORER allows users to store design examples in the catalog without checking for duplications and redundancies. Other systems store only prototypes (Gero, 1990), or prototypes and a small number of examples that are a variation of them (Riesbeck, 1988). These allow users to access *good* examples easily and prevent the chaotic growth of the size of the catalog. However, by not including failure cases, these catalogs prevent users from learning what went wrong in the past.

Many case-based reasoning systems support comprehension and adaptation of cases (Slade, 1991). CATALOGEXPLORER supports the comprehension of examples by allowing users to evaluate them with CONSTRUCTION ANALYZER. Adaptation is done by the users by bringing an example into the *Work Area* in JANUS-CONSTRUCTION. No efforts have been made toward automating adaptation in our approach.

Achievements

In CATALOGEXPLORER, users gradually narrow a catalog space. By analyzing the retrieved information, they can incrementally refine a specification and a construction in JANUS. The retrieval mechanisms described in this paper allow users to access information relevant to the task at hand without forming queries or navigating in information spaces. Use of a partial specification and a partial construction based on a retrieval by reformulation paradigm allows users and the system to share control and responsibility for retrieval.

The system can infer the relevance of subjective hidden features specification and provide users with an explanation for the inferences used. The underlying domain-knowledge can be dynamically derived from the content of the argumentative hypermedia component. The ordering of retrieved examples by the computed appropriateness values support dealing with the problem of partial matching and multiple contradictory features of a design object.

By integrating knowledge-based construction, hypermedia argumentation, catalogs of prestored design objects, and specification components, several crucial design activities can be supported, such as recomputing large information spaces to make them relevant to the task at hand, allowing the situation to talk back, and supporting reflection-in-action (Schoen, 1983).

Limitations

We have not been confronted with the many problems associated with managing large spaces effectively because our design object information spaces (palettes, arguments, catalogs, critics) have been rather small. A lack of mechanisms associating formal representations that can be interpreted by the system with the textual representations used in the argumentative hypermedia component and in the specification component forces us to manually derive the specification-linking rules.

The current specification component needs to be extended and systematized. A task at hand partially articulated by the specification can be used to dynamically determine the set of relevant arguments in the argumentation component. A link between construction and specification can reduce the size of a set of design units displayed in the palette in the construction component by eliminating irrelevant ones. Articulation of the task at hand can be used not only for reducing information spaces but also for guide and constrain design processes. For example, application of specification-linking rules can enable the system to detect inconsistencies between specification and construction.

The representation of design examples in the catalog needs to be enriched both formally and informally. The specification needs to be stored together with the constructed floor layout. More support mechanisms are needed to annotate and add arguments, enabling users to record specification and design rationale associated with a specific design stored in the catalog.

Conclusion

Dealing with ill-defined problems requires the integration of problem setting and problem solving. This implies that the task at hand cannot be fully articulated at the beginning, but only be incrementally refined. The refinement is driven by identifying the most relevant design objects (e.g., parts in the palette, critics in the construction analyzer, arguments in the argumentative component and design examples in the catalog). The power of a design environment is based on the integration within the multifaceted architecture. In this paper we have described mechanisms linking partial specifications and a catalog of prestored designs, thereby making design objects stored in a catalog relevant to the task at hand without forcing users to articulate queries or navigate in information spaces.

Acknowledgments

The authors would like to thank the members of the Human-Computer Communication Group at the University of Colorado, who contributed to the development of the architecture of the multifaceted design environment and instantiated different components of it.

References

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

Fischer, G. 1990. Cooperative Knowledge-Based Design Environments for the Design, Use, and Maintenance of Software, in Software Symposium'90, 2-22, Kyoto, Japan.

Fischer, G., and Lemke, A.C. 1988. Construction Kits and Design Environments: Steps Toward Human Problem-Domain Communication, *Human-Computer Interaction*, 3(3):179-222.

Fischer, G., and Nakakoji, K. 1991. Empowering Designers with Integrated Design Environments, in Proceedings of the First International Conference on Artificial Intelligence in Design, Royal Museum of Scotland, Edinburgh, UK, Forthcoming.

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

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

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

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

Gero, J.S. 1990. Design Prototypes: A Knowledge Representation Schema for Design, *AI Magazine*, 11(4):26-36.

Halasz, F.G. 1988. Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems, *Communications of the ACM*, 31(7), July:836-852.

Hayes, J.R. 1978. *Cognitive Psychology — Thinking and Creating*, Dorsey Press, Homewood, IL.

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

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

Kolodner, J.L. 1990. *What is Case-Based Reasoning?*, In AAAI'90 Tutorial on Case-Based Reasoning, 1-32.

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

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

Norman, D.A. 1986. *Cognitive Engineering*, in D.A. Norman, and S.W. Draper (eds.), *User Centered System Design, New Perspectives on Human-Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, NJ, 31-62, ch. 3.

Patel-Schneider, P.F. 1984. *Small Can Be Beautiful in Knowledge Representation*, AI Technical Report 37, Schlumberger Palo Alto Research.

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

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

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

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

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

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

Simon, H.A. 1973. The Structure of Ill-Structured Problems, *Artificial Intelligence*(4):181-200.

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

Slade, S. 1991. Case-based Reasoning: A Research Paradigm, *AI Magazine*, 12(1), Spring:42-55.

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

Stefik, M.J. 1986. The Next Knowledge Medium, *AI Magazine*, 7(1), Spring:34-46.

Swartout, W.R., and Balzer, R. 1982. On the Inevitable Intertwining of Specification and Implementation, *Communications of the ACM*, 25(7), July:438-439.

Williams, G. 1984. The Apple Macintosh Computer, *BYTE*, 9(2), February:30-54.

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