University of Colorado at Boulder

Department of Computer Science

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

CATALOG EXPLORER: Exploiting the Synergy of Integrated Design Environments

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

[1]Department of Computer Science and Institute of Cognitive Science
Campus Box 430
University of Colorado, Boulder, Colorado 80309

[2]Software Research Associates, Inc., Japan

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

IN PROCEEDINGS OF SOFTWARE SYMPOSIUM '90, KYOTO, JAPAN, JUNE 7 - 8, 1990

Abstract: Reuse is the use of previously acquired concepts and objects in a new situation. With high-functionality software development environments, a sophisticated support mechanism for retrieval of information for reuse is crucial. This paper discusses the retrieval of pre-stored design information in *knowledge-based domain-oriented design environments (KDDEs)*. KDDEs support the entire design process including specification, construction, evaluation, and reuse. In this paper, we focus on a system, CATALOG EXPLORER, which supports retrieval of design examples from the catalog by utilizing a specification. Iterative refinement of the specification and retrieval of design information from the catalog will allow users to reuse design in a natural manner.

Acknowledgements: The authors would like to thank Andreas Lemke, who helped us to develop the ideas behind this paper, and the members of Human Computer Communication group at Computer Science Department of Colorado University, who provided us with valuable comments and suggestions. The research was supported by a grant of Software Research Associates (SRA), Tokyo.

CATALOG EXPLORER: Exploiting the Synergy of Integrated Design Environments

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

[1]Department of Computer Science
University of Colorado, Boulder, Colorado 80309, USA
and
[2]Software Research Associates, Inc., Japan

Abstract

Reuse is the use of previously acquired concepts and objects in a new situation. With high-functionality software development environments, a sophisticated support mechanism for retrieval of information for reuse is crucial. This paper discusses the retrieval of pre-stored design information in *knowledge-based domain-oriented design environments (KDDEs)*. KDDEs support the entire design process including specification, construction, evaluation, and reuse. In this paper, we focus on a system, CATALOG EXPLORER, which supports retrieval of design examples from the catalog by utilizing a specification. Iterative refinement of the specification and retrieval of design information from the catalog will allow users to reuse design in a natural manner.[1]

1. Introduction

Software reuse and redesign have been considered to be of crucial importance for a long time but there has not yet been found a "*silver bullet*" [Brooks 87]. There are a number of obstacles that make software reuse difficult [Fischer 87]. In many software design projects, the strict waterfall model cannot be followed because specifications are developed along with and not prior to the other design phases. Thus, a complete specification is not available for retrieving reusable objects and a system for a reuse must deal with partial, preliminary specifications. Another observation is that we cannot obtain reusable software for free [Nielsen, Richards 89]. A certain amount of additional work is required if we plan to reuse software, which, on the other hand, must not exceed the cost of creation from scratch.

Problems relating to software development include (1) the thin spread of application domain knowledge, (2) fluctuating and conflicting requirements, and (3) communication and coordination breakdowns [Curtis, Krasner, Iscoe 88]. *Knowledge-based domain-oriented design environments (KDDEs)* are an innovative technology that attacks those problems. In KDDEs, application domain knowledge is represented in critics, high-level building blocks, argumentation, and a catalog of complete designs that can be reused for other design problems [Lemke, Fischer 90]. KDDEs provide access to domain knowledge and support learning on demand. The problem of fluctuating and conflicting requirements is addressed by supporting rapid prototyping and the co-evolution of specification and implementation [Swartout, Balzer 82]. The task-oriented building blocks contained in design environments support human problem-domain communication by helping users understand artifacts at the domain level rather than at the code level [Fischer, Lemke 88].

This paper describes a mechanism to support retrieval processes in reuse of design information in KDDEs, and the incremental refinement of specification. In the next section, problems of software reuse are discussed. Then, we describe the basic architecture of KDDEs. In Section 4, we describe CATALOG EXPLORER, which supports retrieval processes by allowing users to retrieve pre-stored design information from the catalog by utilizing *specification*. Finally, we evaluate the system, and discuss future directions.

2. Problems in Reuse of Design Information

Reuse is the use of previously acquired concepts and objects in a new situation. Issues relating to reuse in current software development include at which level reuse should occur and what retrieval mechanism should be used. In this section, we discuss each of these problems.

2.1. Level of Reuse

There are two levels of reuse to consider: the reuse of artifacts and components, and the design rationale of these artifacts [Prieto-Diaz, Freeman 87; Fischer 87]. Currently, software reuse only implies the former, that is, reuse of code. However, the latter is also important because ideas and knowledge of previous design episodes can provide guidance in the future, offer alternatives, and warn of pitfalls. People can use previous work as cases constituting a detailed problem solution, and also can take advantage of them by using them to justify a solution and explain its rationale [Rissland, Skalak 89].

The difficulty with reuse design rationale is that it is not usually recorded physically. It is only available as a body of experience held by software designers who participated

[1]The paper by Gerhard Fischer in these proceedings provides the general context for this work.

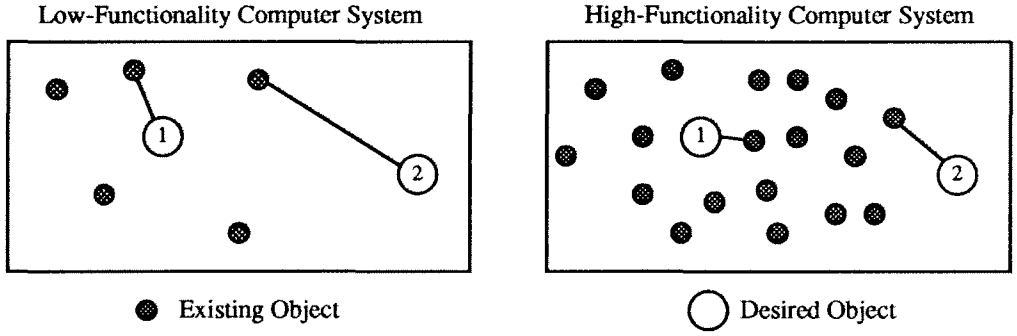

Figure 1: Trade-off Between Accessibility and Usefulness

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

in earlier design projects. The potential to reuse this information is dependent on the designers' ability to retrieve and determine the relevance of previous design efforts. Therefore, to make software reuse effective, both code and design rationale used in the design should be explicitly recorded.

2.2. Retrieval Mechanisms

As for retrieval of information, there is a trade-off between accessibility and usefulness (Figure 1) [Fischer, Girgensohn 90]. With a high-functionality system, the amount of information can be pretty large, and the potential for finding a closer objects to what is needed is high. However, with such a large amount of information, it is impossible to examine each component one by one [Nielsen, Richards 89]. On the other hand, a small information space is easier to search, but it is less useful because the probability that a suitable reusable object exists in the information space is low.

To search a large catalog of reusable objects, a highly specific query must be formulated. However, this is often not possible because a complete specification is not available at the start of a project. Instead, specifications are developed during the course of the project along with the emerging design [Fischer 90].

In an empirical study of *human-human cooperative problem solving* in a hardware store [Reeves 90], we have observed that people do not initially articulate a complete query. Instead, they start from a partial specification and refine it incrementally. In the study, the retrieval process was observed to be a cyclic process of specification refinement and feedback from examples: First a customer describes a partial specification. Then a salesperson provides examples. Then the customer refines the specification by using the given examples, and the salesperson provides the next refined solution. This iterative process continues until a solution is reached. This study indicates that a mechanism is needed that helps users incrementally refine their specification and that provides examples from the catalog.

3. Conceptual Framework

In this section, we first describe the architecture of a KDDE, which supports the entire software development process. Then we discuss the reuse of design and specification in this architecture. We especially focus on how a specification component can help users in retrieving design information from a catalog.

3.1. Knowledge-Based Domain-Oriented Design Environments (KDDEs)

A KDDE is an integrated environment supporting the entire software development process including reuse. A KDDE consists of the following five components as illustrated in Figure 2 [Fischer 90].

- **A construction kit** provides a palette of domain abstractions. It supports the construction of artifacts by allowing users to assemble a solution from design components by direct manipulation and other interaction styles.
- **An argumentative hypertext system** captures the design rationale. It makes a user aware of issues relevant to the construction situation, possible answers, and argumentation by utilizing the system's domain knowledge.
- **A specification component** allows users to explicitly specify their requirements to the system. It is expected to be modified and augmented during an entire design process, rather than being fully defined and prepared before starting designing.
- **A simulation component** allows a user to simulate the artifact being constructed and helps the situation talk back to the user [Schoen 83].
- **A catalog** consists of a collection of pre-stored artifacts illustrating the space of possible designs in the domain. Each component of the catalog is a design example incorporating its specification, design rationale, and decision making process as well as the concrete artifact.

The use of each of these system components is augmented

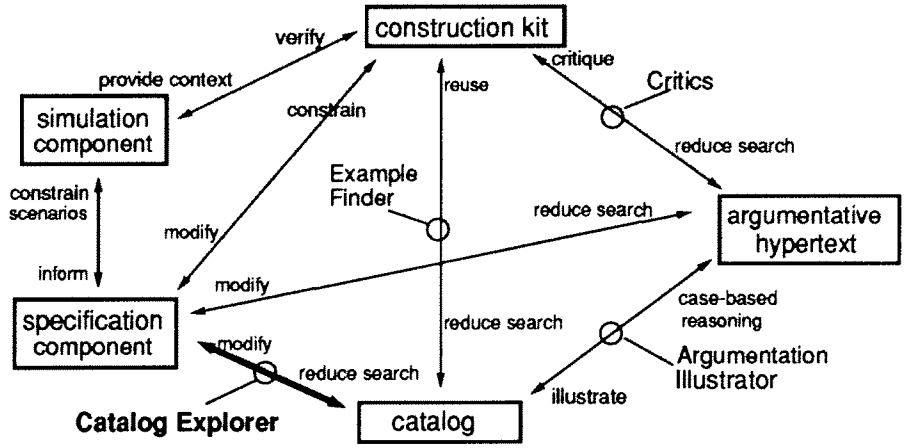

Figure 2: A Multifaceted Architecture for KDDEs

This figure shows the components of the multifaceted architecture. It stresses the integration between the different components by indicating the prototype systems which link the individual components with each other. The CATALOG EXPLORER is described in this paper, the EXAMPLE FINDER in [Shinmori 90], the CRITICS and the ARGUMENTATION ILLUSTRATOR in [Fischer 90].

by other components in the architecture. Thus, these components form a synergistic whole. As users go back and forth among these components, the problem space is narrowed and all facets of the artifact including its specification and design rationale are refined.

We have developed several systems using this architecture. Some links among the components in Figure 2 have been realized in these systems. The first three have been implemented in a system for kitchen design, the fourth one in the domain of software design.

- Knowledge-based critics link construction to argumentation by pointing out suboptimal features of the artifact and accessing the corresponding issue in the argumentative hypertext component.
- Argumentative hypertext and the catalog are linked by a facility that selects examples from the catalog that illustrate an issue.
- CATALOG EXPLORER links specification to the catalog by finding examples that satisfy the specification.
- EXAMPLE FINDER [Shinmori 90] links from construction to catalog by finding examples that are relevant to the artifact being constructed.

3.2. Reuse and Specification in KDDEs

The catalog supports reuse by providing solutions to similar design problems. Design examples in a catalog can also be used to illustrate argumentation by providing specific examples for issues, answers, and arguments. The specification constrains what artifacts can be created in the construction kit. The specification also filters the argumentative hypertext and the set of reusable objects available in the catalog.

In the rest of the paper, we discuss the CATALOG EXPLORER, which supports the link between the specification component and the catalog in Figure 2.

4. CATALOG EXPLORER

4.1. JANUS and CATALOG EXPLORER

CATALOG EXPLORER helps users reduce the search space of catalog examples by utilizing the information provided by a specification component. The system is a subcomponent of the JANUS system [Fischer, McCall, Morch 89a], which has been developed based on the architecture described in Section 3.1. The domain of JANUS is kitchen design. The system is implemented in Common Lisp, and runs on Symbolics Lisp machines. JANUS consists of three subsystems: (1) JANUS-CRACK [Fischer, Morch 88], which is a construction kit; (2) JANUS-VIEWPOINTS, an argumentative hypertext system [Fischer, McCall, Morch 89b]; (3) CATALOG EXPLORER, which supports catalog search.

CATALOG EXPLORER operates in the following manner:

- It helps users reduce the size of the catalog by selecting only the designs which obey the partial specification.
- It helps users to incrementally develop a specification by using the catalog as a source of ideas.

The system searches for design information by communicating with users via a specification. After searching the catalog and finding reusable design information, users can modify and utilize the information in both the construction kit and argumentative hypertext system. In the former, users are allowed to design by modifying the retrieved result instead of designing from scratch. In the latter, the argumentation utilizes design examples as a means of supporting the arguments. Users may modify their specification after having created a new design situation.

In Section 4.2, we describe the representation of catalog examples in CATALOG EXPLORER. Then, in Section 4.3,

we describe the retrieval process of reuse in CATALOG EXPLORER.

4.2. Representation of Examples in the Catalog in CATALOG EXPLORER

In using the catalog, users should be able to search for reusable design artifacts as well as design concepts. Thus, the information such as how requirements and design decisions are made, represented, communicated, and changed, as well as how these decisions impacted subsequent development processes, must be recorded and distributed in the KDDE.

The presumption made is that design examples have already been constructed and stored in the catalog. CATALOG EXPLORER is based on the HELGON system [Fischer, Nieper-Lemke 89], and the design examples are stored as data in a KANDOR knowledge base [Patel-Schneider 84].

Each catalog example consists of a kitchen floor layout and a design specification including functional and social features. In the knowledge base, each kitchen example is represented as a kitchen design instance. Each kitchen design example in the catalog has a set of slots including *General Attributes*, *Floor Layouts*, *Physical Features*, and *Functional/Social Features*. *General Attributes* incorporates general information for a design example. *Floor Layouts* is pictorial data, namely a concrete design artifact. *Physical Features* describe physical characteristics for each design example. Finally, *Functional/Social Features* describe a design example in a more abstract manner. The two slots of them, *Good-For* and *Bad-For*, are filled with some of the following values: *Small-children*, *Often-entertain*, *Enjoy-cooking*, *Safety*, or *Efficiency*. These features play an important role in forming queries from requirement specifications. This is discussed in Section 4.3.

By the nature of the KANDOR knowledge representation mechanism [Patel-Schneider 84], kitchen examples are automatically classified according to its features. For instance, if an example has an *L-shape* property for the *shape* attribute, then that kitchen is implicitly classified as a member of the *L-shape* kitchen class. There is no hierarchical relationship between classes. Kitchen examples are not classified exclusively. One kitchen example can be classified as a member of multiple kitchen classes.

4.3. Retrieval Process in CATALOG EXPLORER

The characteristics of the retrieval process in CATALOG EXPLORER are as follows:

1. CATALOG EXPLORER provides a specification sheet and formulates a query from the specified requirements.
2. CATALOG EXPLORER supports retrieval of design information from the catalog, based on the retrieval by reformulation paradigm [Williams 84; Fischer, Nieper-Lemke 89].
3. CATALOG EXPLORER evaluates retrieved design information using the system's domain knowledge.

In the rest of this subsection, we describe each step in more detail.

Retrieval by Specification: The specification in JANUS includes concepts such as: *designing a kitchen for a large family*, or *the kitchen is appropriate for frequent entertainment*. This type of information typically plays an important role in the use of the catalog in an actual kitchen design situation.

Such information can often be very arbitrary. However, analyzing the initial questionnaires that professional kitchen designers are using, we have found that different questionnaires share many terms and representations, and that the vocabulary used in these questionnaires is limited. Interestingly, this limited vocabulary seems to be sufficient to cover the arguments stored in the issue base in JANUS-VIEWPOINTS. This observation suggests that a menu-based specification sheet can be used in the specification component.

In specifying requirements in CATALOG EXPLORER, users are permitted to use only a limited vocabulary by using a simple menu-based specification sheet. In the specification sheet, users can explicitly specify a *Do-not-care* value for a slot, in which case, there will be no restriction for that slot. The specification sheet used in CATALOG EXPLORER is shown in Figure 3.

The requirements are related to *functional/social feature* slots for each design example, and CATALOG EXPLORER will form a query by interpreting the specified requirements. For example, if a user answers "*Do you often give parties?*" with *Yes*, the query "*Good-for: Often-Entertain*" and "*Shape: Corridor*" will be generated. The second query will be generated because the following issue-answer-argument pair is stored in the issue-base: "*If you often give parties, a corridor-shape kitchen is preferable because it provides an efficient work flow for more than one cook.*" In this manner, the system's domain knowledge helps users form a query from the information given by the specification component.

Retrieval by Reformulation: The paradigm of retrieval by reformulation was derived from a psychological theory of human remembering [Williams 84]. It allows users to incrementally improve a query by critiquing the results of previous queries. This paradigm serves as a mechanism to allow users to iteratively search for more appropriate design information and to refine their specification, rather than being constraint to the query that is initially generated by the system [Fischer, Henninger, Redmiles 90].

Evaluation of retrieved results: Retrieved results can be critiqued and praised by using the system's domain knowledge stored in JANUS-VIEWPOINTS. This helps users to evaluate and select the retrieved results.

In addition to these main features, users of CATALOG EXPLORER can mark a matching item with a bookmark, thus personalizing the catalog.

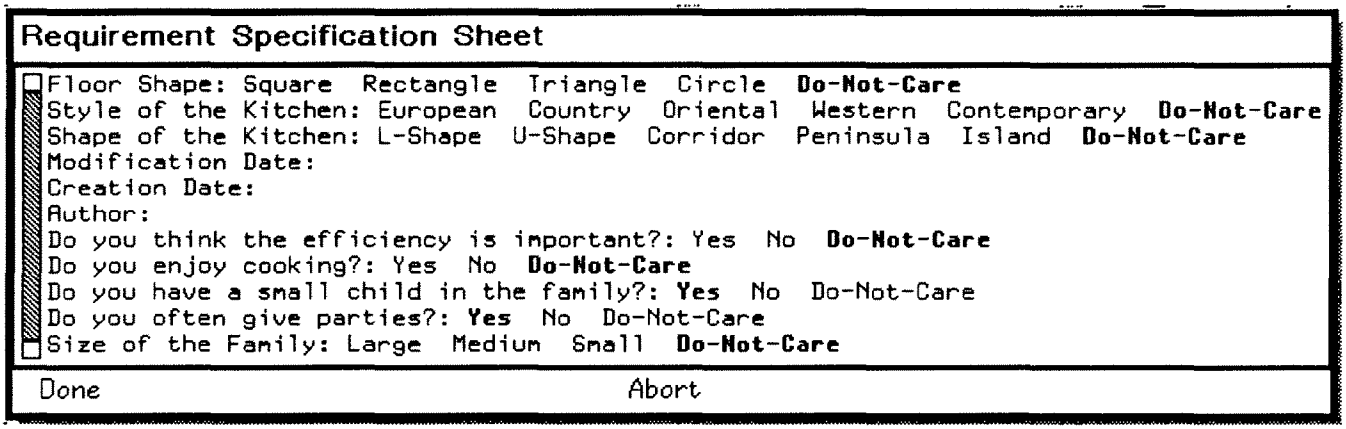

Figure 3: Specification Sheet in CATALOG EXPLORER

4.4. Scenario

Figure 4 shows a screen image of CATALOG EXPLORER. The *Category Hierarchy* window shows the hierarchical structure of the database. The *Query* window shows a query formed either by the system or by a user. The *Matching Design* window lists all matching design examples in the catalog, and the right half of the screen shows one of those matching examples. Finally, the *Bookmarks of Design* window can be used to personalize the catalog.

Here is a simple scenario using CATALOG EXPLORER. A user wants to design a kitchen for his family by using JANUS, and is operating in JANUS-CRACK (a construction kit). He has decided to reuse a design stored in the catalog. He invokes the *Catalog* command in JANUS-CRACK, and activates CATALOG EXPLORER.

Initially, dozens of design examples are available to him in CATALOG EXPLORER. He invokes the *Specify* command, and the requirement specification sheet appears on the screen. Since his family consists of his wife, a 3-year old boy, and himself, and they like to have parties, he answers both "*Do you have a small child in the family?*" and "*Do you often give parties?*" with *Yes*, and leaves the other fields as they are. After he enters *Retrieve from Spec*, the query is formed by CATALOG EXPLORER, and now four kitchen design examples satisfying the query are retrieved. Figure 4 shows the result of this retrieval. If he likes the design, then he can add that design to the list of bookmarks via the *Add to Bookmark* command.

5. Discussion of CATALOG EXPLORER

CATALOG EXPLORER relieves users of having to examine all catalog examples one by one. Users even do not have to form a structured query to retrieve the necessary information. All they have to do is specifying their requirements in the specification sheet. An iterative refinement of the specification is allowed by the retrieval by reformulation paradigm. Thus, a smooth transition between a catalog and a specification component is provided by the system.

Currently, the design rationale associated with each catalog example in CATALOG EXPLORER is low level and is presumed to have already been constructed. For practical use, more complex information should be attached to design examples. In addition, users of JANUS should be allowed to construct specifications in terms of the constructive situation so that the design rationale of a constructed design could be accessible by the system later.

Users of CATALOG EXPLORER often do not understand "*Why this example is retrieved.*" The system should *understand* the features and should be able to provide an explanation to users if needed. This could be achieved by supporting a link between an argumentative hypertext system to the catalog as discussed in Section 6.

6. Future Directions

In this section, we discuss directions for future work on CATALOG EXPLORER.

Support for Other Transition Links: Specification can be used to determine the set of relevant arguments in JANUS-VIEWPOINTS. Also, the link between construction kit and specification can be supported by limiting the available design components based on the specification, and finding inconsistencies between the constructive situation and the specification. The domain knowledge provided by an argumentative hypertext system can be used to reason about catalog examples. A simulation component should also be added to JANUS. Thus, the entire design process could be supported by one environment.

Ordering of Retrieved Examples: One way to make the result of retrieval processes more comprehensible is to order the retrieved design examples. There could be a preference list among all requirements, and those examples which satisfy higher preference requirements would be retrieved first. Those which satisfy low priority requirements would be retrieved last.

This capability would make CATALOG EXPLORER a more flexible and adaptive system. Also, a more sophisticated

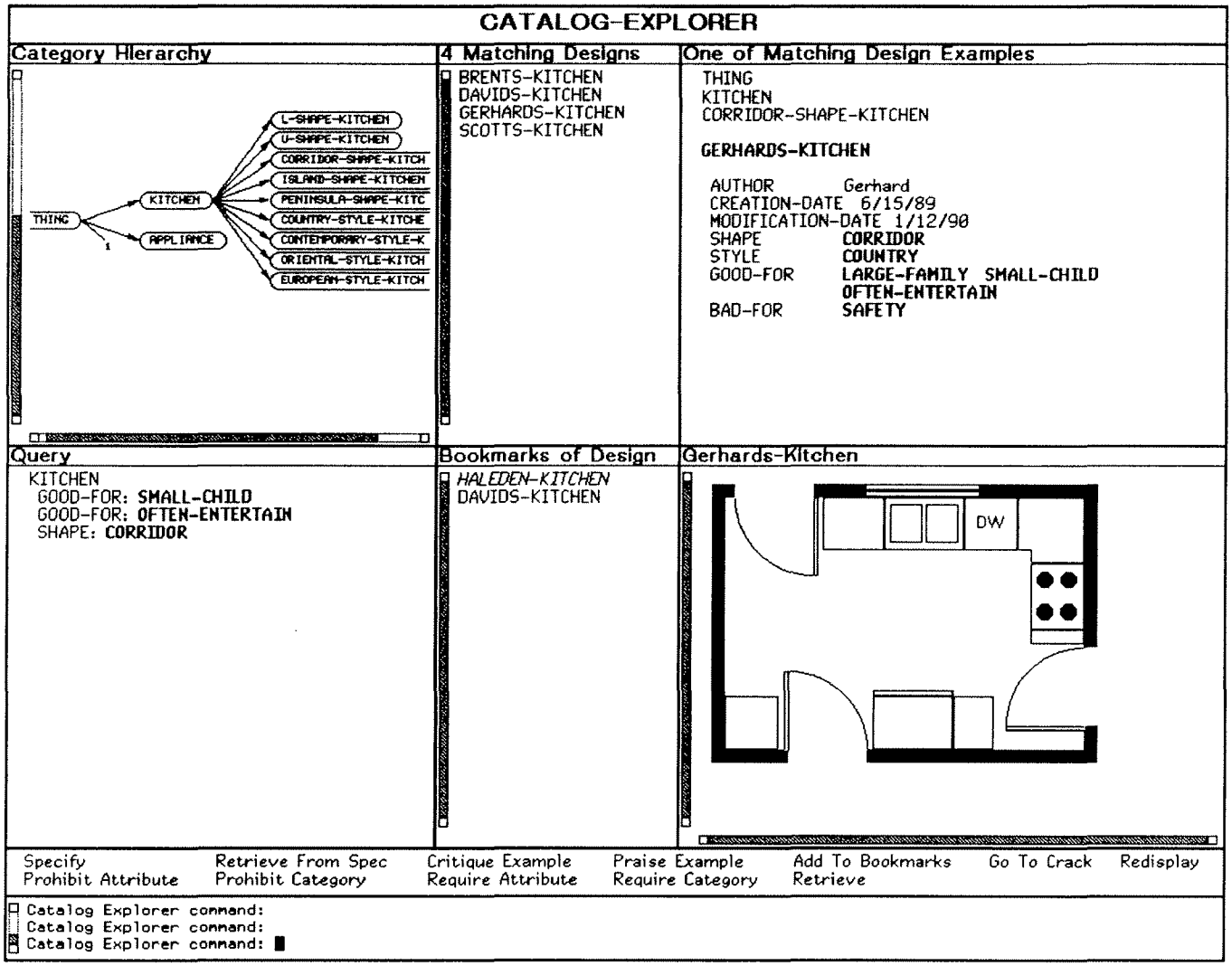

Figure 4: Screen Image of CATALOG EXPLORER

CATALOG EXPLORER generates the query from a user's specification and retrieves matching design examples based on that query. In the figure, four matching design examples are provided to a user. One of them (*Gerhards-Kitchen*) is shown on the screen as a concrete example. A user may refine their specification after examining some retrieved examples.

multiple specification paradigm could be realized in the sense that users could modify their queries without changing a set of required categories but by changing weights of each category. Then, they could compare the retrieved results with slightly different weighs with their requirements.

Contradictory Features: It sometimes occurs that one design example has multiple contradictory features. For example, there can be a kitchen design example such that the location of a stove in it is *good for large-family*, but the location of refrigerator in it is *bad for large-family*. This could be solved by the idea of weighted links between a particular design example and contradictory features [Henninger 90].

Level of Assembly: Another issue is how to manage the level of assembly of design components in the catalog. There seems to be no answer for what is the appropriate level of configuration. Sometimes users do not need a whole kitchen example but are only interested in looking at a part of a kitchen, for example, a *cleanup center*. There are several alternatives to handle this problem. One of them is to extract a part of a design from a complete one, or the catalog should incorporate parts of design assemblies as well as complete kitchen designs.

7. Conclusion

By using CATALOG EXPLORER, users can search a large catalog space in a natural manner. Incremental refinement of a specification together with retrieval by reformulation are a sophisticated retrieval methodology in the use of a catalog of reusable components.

Given appropriate design examples from the catalog, users can modify them in a construction kit (JANUS-CRACK), instead of designing a new design from scratch. Designers can also use the examples in the catalog for reasoning and

helping to understand arguments provided by an argumentative hypertext component (JANUS-VIEWPOINTS). Thus, reuse of design information can be realized as a part of the entire design process in a KDDE.

Design activities are complex and incorporate many cognitive issues, such as recognizing a problem, finding strategies, understanding given information, and adapting the information to the situation. We have chosen the simple kitchen domain because issues related to building design support systems are more explicit in this domain and because the domain is easy to understand. We believe that the same problems occur in software design and that the results of our research are applicable in the software domain as well.

Acknowledgments

The authors would like to thank Andreas Lemke, who helped us to develop the ideas behind this paper, and the members of Human Computer Communication group at Computer Science Department of Colorado University, who provided us with valuable comments and suggestions. The research was supported by a grant of Software Research Associates (SRA), Tokyo.

References

[Brooks 87]
F.P. Brooks Jr., *No Silver Bullet, Essence and Accidents of Software Engineering*, IEEE Computer, Vol. 20, No. 4, April 1987, pp. 10-19.

[Curtis, Krasner, Iscoe 88]
B. Curtis, H. Krasner, N. Iscoe, *A Field Study of the Software Design Process for Large Systems*, CACM, Vol. 31, No. 11, November 1988, pp. 1268-1287.

[Fischer 87]
G. Fischer, *Cognitive View of Reuse and Redesign*, IEEE Software, Special Issue on Reusability, Vol. 4, No. 4, July 1987, pp. 60-72.

[Fischer 90]
G. Fischer, *Cooperative Knowledge-Based Design Environments for the Design, Use, and Maintenance of Software*, Software Symposium'90, Kyoto, Japan, 1990.

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

[Fischer, Henninger, Redmiles 90]
G. Fischer, S. Henninger, D. Redmiles, *A Conceptual Framework and Innovative Systems for Accessing Knowledge for Software Reuse*, Technical Report, Department of Computer Science, University of Colorado, Boulder, CO, 1990.

[Fischer, Lemke 88]
G. Fischer, A.C. Lemke, *Construction Kits and Design Environments: Steps Toward Human Problem-Domain Communication*, Human-Computer Interaction, Vol. 3, No. 3, 1988, pp. 179-222.

[Fischer, McCall, Morch 89a]
G. Fischer, R. McCall, A. Morch, *Design Environments for Constructive and Argumentative Design*, Human Factors in Computing Systems, CHI'89 Conference Proceedings (Austin, TX), ACM, New York, May 1989, pp. 269-275.

[Fischer, McCall, Morch 89b]
G. Fischer, R. McCall, A. Morch, *JANUS: Integrating Hypertext with a Knowledge-Based Design Environment*, Proceedings of Hypertext'89, ACM, New York, November 1989, pp. 105-117.

[Fischer, Morch 88]
G. Fischer, A. Morch, *CRACK: A Critiquing Approach to Cooperative Kitchen Design*, Proceedings of the International Conference on Intelligent Tutoring Systems (Montreal, Canada), ACM, New York, June 1988, pp. 176-185.

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

[Henninger 90]
S. Henninger, *Defining the Roles of Humans and Computers in Cooperative Problem Solving Systems for Information Retrieval*, Proceedings of the AAAI Spring Symposium Workshop on Knowledge-Based Human Computer Communication, March 1990.

[Lemke, Fischer 90]
A.C. Lemke, G. Fischer, *A Cooperative Problem Solving System for User Interface Design*, Proceedings of AAAI-90, Ninth National Conference of Artificial Intelligence, 1990.

[Nielsen, Richards 89]
J. Nielsen, J.T. Richards, *The Experience of Learning and Using Smalltalk*, IEEE Software, May 1989, pp. 73-77.

[Patel-Schneider 84]
P.F. Patel-Schneider, *Small Can Be Beautiful in Knowledge Representation*, AI Technical Report 37, Schlumberger Palo Alto Research, October 1984.

[Prieto-Diaz, Freeman 87]
R. Prieto-Diaz, P. Freeman, *Classifying Software for Reusability*, IEEE Software, January 1987, pp. 6-16.

[Reeves 90]
B. Reeves, *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, 1990, forthcoming.

[Rissland, Skalak 89]
E.L. Rissland, D.B. Skalak, *Combining Case-Based and Rule-Based Reasoning: A Heuristic Approach*, Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, 1989, pp. 524-530.

[Schoen 83]
D.A. Schoen, *The Reflective Practitioner: How Professionals Think in Action*, Basic Books, New York, 1983.

[Shinmori 90]
A. Shinmori, *Finding and Reusing an Appropriate Example for User Interface Program Design*, Unpublished Master's Thesis, University of Colorado, Boulder, May 1990.

[Swartout, Balzer 82]
W.R. Swartout, R. Balzer, *On the Inevitable Intertwining of Specification and Implementation*, Communications of the ACM, Vol. 25, No. 7, July 1982, pp. 438-439.

[Williams 84]
M.D. Williams, *What Makes RABBIT Run?*, International Journal of Man-Machine Studies, Vol. 21, 1984, pp. 333-352.