Software Reuse in Integrated, Domain-Oriented Knowledge-based Design Environments

Kumiyo Nakakoji
Department of Computer Science and Institute of Cognitive Scien
University of Colorado
Boulder, Colorado 80309, USA

E-mail: kumiyo@cs.colorado.edu

Software Engineering Laboratory
Software Research Associates, Inc.
1-1-1 Hirakawa-cho, Chiyoda-ku, Tokyo 102, Japan

Abstract

Our approach to cope with ill-structured software design problems is to empower software designers with domain-oriented, knowledge-based software environments that support coevolution of requirements specification and design construction. The synergy of this integration supports designers in reusing prestored design objects relevant to their current task at hand as articulated with a partial specification and a partial construction.

Keywords: ill-structured software design problems, knowledge-based software environments, multifaceted architecture, integration of specification and construction, reuse and redesign, information retrieval, case-based reasoning

1 Introduction

Many software design problems are ill-structured [Simo 73] with fluctuating and conflicting requirements. Traditional software design, use and maintenance methodologies are inadequate for dealing with ill-structured problems. Empirical studies have shown that those problems are best understood as design problems in which the design space and requirements unfold incrementally [Fisc 91d]. These problems call for an integrated approach, supporting coevolution of requirements specification and design construction.

Complex systems are not designed from scratch, but they evolve [Dawk 87]. Complex systems evolve faster if they can be build on stable subsystems [Simo 81]. Previously solved design problems

and design experiences should be used for informing designers of possible solutions, failures, or justifications of current decisions.

The above requirements have led us to conclude that software should be developed in integrated, domain-oriented, knowledge-based design environments that support the whole lifecycle of software – requirement articulation, design, maintenance, and reuse. Our goal is to empower software designers and to augment their capability and productivity through artificial intelligence and human-computer interaction techniques.

The research described here addresses issues of software reuse in such design environments. Design environments store previously constructed design solutions as domain knowledge. By integrating design creation tools and domain knowledge bases, our environments based on the multifaceted architecture support the cycle of *location*, *comprehension*, and *modification* of prestored design objects in one coherent substrate. One system component, CATALOGEXPLORER, which supports the location phase in design environments, is described.

2 Problems of Current Approach for Software Reuse

Reuse in software development should be supported in such a way that designers can apply previously made design efforts not only to a final design artifact but also to a design process and rationale behind the artifact. Truly reusable design objects can be achieved only when those objects have been created in the same environment because then a system could store necessary information along with the development of objects for later reuse [Frak 90]. Traditional CASE tools fail because they do not directly support reusing created design artifacts [Curt 91]. Reuse should be supported in the same context as the development of the software takes place.

The richer the information spaces are, the more expensive it is to access, in terms of both computational and cognitive costs. Designers first must locate information, then comprehend the retrieved information, and modify it according to their current needs. Comprehension may directly lead to further retrieval, modification may require further comprehension. Software designers should not be distracted by the information access, which is not their primary concern in designing software.

Conventional query-based retrieval mechanisms break down [Fisc 91c] because they assume that humans can articulate what they are looking for by formulating a highly specific query. Stored software objects in most software reuse support systems are represented in terms of a solution domain; namely, final design solutions. When software designers need those information, they usually have in their mind application goals, or requirements, which are represented in a problem domain, but not concrete solutions. Therefore, they cannot form a specific query for retrieving such stored objects.

A purely *navigational access* provided by a browsing mechanism only partially solves the above problem because it still requires that the information space has a fairly rigid and predetermined structure. For dealing with ill-defined domains there is not one *right* structure for the information space and the structure needs to be tailored according to the task at hand. Moreover, humans may get lost while tracing links among the information spaces if the search space is large and the structure is complex [Hala 88].

3 The Approach

To this end, we have developed a conceptual framework incorporating evolutionary design and maintenance of software. The conceptual framework is based on cooperative problem solving between designers and an integrated, knowledge-based, design environment, which supports the coevolution of specification and construction [Ritt 84, Swar 82]. The design environment framework is instantiated by innovative system building efforts supporting information access in high-functionality software environments, providing feedback on partial designs, and enhancing reuse, redesign and end-user modifiability. We have developed a multifaceted architecture for such environments, consisting of design creation tools that support both requirement specification and design construction, and domain knowledge bases (see Figure 1).

Orthogonal to the elements of an environment, the architecture supports three basic processes for information access: *location*, *comprehension*, and *modification* of stored design objects. Reuse of design objects is supported by using the catalog base. The catalog allows designers to reuse various perspectives of design objects and design experiences in the domain, either of their own, or of others that used the same design environments.

The multifaceted architecture enables an innovative information retrieval technique for compli-

menting the conventional approaches in locating design objects stored in the environments. The architecture takes advantage of the synergistic integration that conventional information retrieval systems are lacking. A partially articulated task at hand can be used to filter out irrelevant information and reduce related information spaces. It relieves designers of the task of specifying queries or navigating in information spaces for retrieval, thereby supporting designers to retain their working context. The environments then support the designers to comprehend the retrieved information, which may lead to refinement of their specification and construction.

4 CATALOGEXPLORER

CATALOGEXPLORER is a system component of the integrated design environments based on the multifaceted architecture. CATALOGEXPLORER illustrates how a designer is supported in locating prestored design object in the example domains of plotting data in graphic programming [Fisc 91b].

The CATALOGEXPLORER augments the frame-based search technique provided by the HELGON system [Fisc 89] with the following mechanisms for retrieving from the catalog, design objects relevant to the task at hand:

- ordering design examples by computed appropriateness values based on the current specification (*retrieval from specification*).
- analyzing the current construction and retrieving similar examples from the catalog (*retrieval from construction*).

Retrieval from Specification. CATALOGEXPLORER provides a questionnaire type specification component that allows designers to specify requirements to their current task, at the level of a problem domain. These requirements relate to *hidden features* of a design. Hidden features are related to functions of the design rather than low-level or surface structure [Gero 90]. For retrieving design objects by hidden feature specifications, the system must have the domain knowledge for mapping those features onto the surface structure. Such domain knowledge is represented by:

1. *specification-linking rules* that link each subjective hidden feature specification item to a set of surface condition rules (in the integrated environment, this domain knowledge can be derived from the content of the argumentation base; see Figure 2).
2. a metric that computes an *appropriateness* value of each design example in the catalog in terms of a partial specification for dealing with trade-offs among contradictory specifications (the system then reorders the examples according to those computed values).

For example, suppose a designer wants to create a program to plot the results of two persons playing one-on-one basketball. The designer first specifies some of the requirements using the specification sheet provided by CATALOGEXPLORER, such as "*need to illustrate the correlation of the two values*" (see Figure 2). The system uses domain knowledge in the form of a specification rule, "*illustrating correlation requires horizontal and vertical axes and a diagonal line.*" Using

this knowledge, the system searches the stored example programs in the catalog which have those characteristics. The search yields as the most appropriate one, for instance, an example program that plots the results of two people playing squash.

Retrieval from Construction. For retrieving design examples relevant to a partial construction, one must deal with the issues of matching design examples in terms of surface features of a design, namely, at a structural level. Two *domain-specific parsers* analyze the design under construction by articulating types of design components being used at LISP structure level, and at graphics semantic level. Then the system retrieves design examples from the catalog each of which has the same set of types of design components in one of the two structure, according to a users' request.

5 Related Work

The primary uniqueness of our work is our emphasis on "*human-centeredness.*" For example, the LaSSIE project [Deva 91] overlaps with our approach in using a knowledge-base for supporting software reuse. However, while the LaSSIE project stresses the knowledge base foremost in terms of its structure, representation, and access methods, we stress capturing the user's task at hand by supporting requirement specification and design construction concurrently, and retrieving pre-stored design objects in terms of that task. Several software reuse systems maintain representations of what I have referred to as the higher-level requirement specification. They, however, use formal, automated techniques to produce new programs and do not support humans thinking in a natural manner [Neig 84, Reub 90]. As designers' understanding of potential reuse components and examples increases, they are able to modify their requirements and the solution design accordingly

[Fisc 91c, Scho 83]. From the information retrieval perspective, searching a catalog in the integrated design environment raises many issues in common with retrieval in case-based reasoning. Our approach addresses an indexing problem [Kolo 90] by combining abstract and surface features, using the specification-linking rules that support analogical matching that is similar to *systematicity-based matching* [Navi 88].

References

[Curt 91] B. Curtis, "The Psychology of Software Development", Tutorial Presented at Conference on Human Factors in Computing Systems, CHI'91 (New Orleans, LA)

[Dawk 87] R. Dawkins, *The Blind Watchmaker*, W.W. Norton and Company, New York - London, 1987.

[Deva 91] P. Devanbu, R.J. Brachman, P.G. Sefridge, B.W. Ballard, "LaSSIE: A Knowledge-Based Software Information System", *Communications of the ACM*, Vol. 34, No. 5, 1991, pp. 34-49.

[Fisc 91a] G. Fischer, A.C. Lemke, R. McCall, A. Morch, "Making Argumentation Serve Design", *Human Computer Interaction*, 1991, (in press)

[Fisc 91b] G. Fischer, A. Girgensohn, K. Nakakoji, D. Redmiles, "Supporting Software Designers with Integrated, Domain-Oriented Environments", *IEEE Transactions on Sofware Engineering, Special Issue on "Knowledge Representation and Reasoning in Software Engineering"*, 1991, (submitted)

[Fisc 91c] G. Fischer, S.R. Henninger, D.F. Redmiles, "Cognitive Tools for Locating and Comprehending Software Objects for Reuse", *Thirteenth International Conference on Software Engineering (Austin, TX)*, IEEE Computer Society Press, ACM, IEEE, Los Alamitos, CA, 1991, pp. 318-328.

[Fisc 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.

[Fisc 91d] G. Fischer, B.N. Reeves, "Beyond Intelligent Interfaces: Exploring, Analyzing and Creating Success Models of Cooperative Problem Solving", *Applied Intelligence, Special Issue Intelligent Interfaces*, 1991, (in press)

[Frak 90] W.B. Frakes, P.B. Gandel, "Representing Reusable Software", *Information and Software Technology*, Vol. 32, No. 10, December 1990, pp. 653-664.

[Gero 90] J.S. Gero, "A Locus for Knowledge-Based Systems in CAAD Education," in *The Electronic Design Studio*, M. McCullough et al., eds., Cambridge, MA: The MIT Press, 1990, pp. 49-60.

[Hala 88] F.G. Halasz, "Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems", *Communications of the ACM*, Vol. 31, No. 7, July 1988, pp. 836-852.

[Kolo 90] J.L. Kolodner, "What is Case-Based Reasoning?", In AAAI'90 Tutorial on Case-Based Reasoning, pp. 1-32

[Navi 88] D. Navinchandra, "Case-Based Reasoning in CYCLOPS", *Proceedings: Case-Based Reasoning Workshop*, J. Kolodner,ed., Morgan Kaufmann Publishers, Clearwater Beach, FL, May 1988, pp. 286-301.

[Neig 84] J.M. Neighbors, "The Draco Approach to Constructing Software from Reusable Components", *IEEE Transactions on Software Engineering*, Vol. SE-10, No. 5, September 1984, pp. 564-574.

[Reub 90] H.B. Reubenstein, "Automated Acquisition of Evolving Informal Descriptions", Tech. report, MIT, 1990.

[Ritt 84] H.W.J. Rittel, "Second-Generation Design Methods," in *Developments in Design Methodology*, N. Cross, ed., New York: John Wiley & Sons, 1984, pp. 317-327.

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

[Simo 73] H.A. Simon, "The Structure of Ill-Structured Problems", *Artificial Intelligence*, No. 4, 1973, pp. 181-200.

[Simo 81] H.A. Simon, *The Sciences of the Artificial*, The MIT Press, Cambridge, MA, 1981.

[Swar 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.

6 About the Author

Kumiyo Nakakoji received her B.A. degree in computer science from Osaka University, Japan, in 1986 and her M.S. degree in computer science from the University of Colorado, Boulder, in 1990. She is currently a Ph.D. student in the Department of Computer Science and the Institute of Cognitive Science at University of Colorado, Boulder. Her studies are sponsored through a scholarship from Software Research Associates, Inc., Japan, where she has been an employee since 1986. Ms. Nakakoji is a member of Human Computer Communication group led by Prof. Gerhard Fischer. Her research interests include human-computer communication, artificial intelligence, knowledge-based design environments, case-based reasoning, and software design. Her dissertation title is "Delivering Case-based Information in Knowledge-based Design Environments."