Reusing Software Design : A Generic Architecture-based Approach

Sanjay Bhansali

Knowledge Systems Laboratory, Stanford University
701 Welch Road, Building C, Palo Alto, CA 94304
Tel: (512)-331-3541,(415)-723-8387
Email: bhansali@austin.slcs.slb.com, bhansali@hpp.stanford.edu

Abstract

We are attempting to build a framework and computational environment to assist software designers in designing software systems efficiently and reliably. Our approach is based on abstracting the design as well as the process involved in designing a software system and reusing them to design new systems. The emphasis is on building an environment in which the system acts as an assistant that delivers the relevant knowledge to a designer so that he or she may make informed decisions. Our work may be characterized as a semi-formal approach that aims to create a system that would be usable by software designers who are not well-versed with formal methods.

Keywords: Reuse, software design, domain analysis, software process.

Workshop Goals: Networking and exchanging ideas with other researchers in similar areas, identifying problems of practical as well as theoretical interest in software reuse.

Working Groups: Domain analysis/engineering, Tools and environments, Reuse process models.

1 Background

I am interested in the application of novel techniques and tools for improving the efficiency and quality of software development through reuse. My earlier research in this area was concerned with issues of reuse in the context of program synthesis. Specifically, I investigated how a program synthesizing system could be made more efficient by reusing the derivation history of analogous programs that had been synthesized earlier[1]. Subequently, I have become interested in issues involved in the design of large software systems. I am currently co-leading the KASE (Knowledge Assisted Software Engineering) project at the Knowledge Systems Laboratory, Stanford which is engaged in building a knowledge-based environment that can assist software designers to design systems by reusing generic software designs represented as generic architectures [2, 3].

I am also interested in software processes and issues concerning the reuse of processes in a process-driven environment. I have been involved (in collaboration with Pankaj Garg at Hewlett-Packard Laboratory) in the design and implementation of a system that can support the automated generation of process fragments by generalizing a sequence of user activities in a work session [4].

2 Position

One of the most effective principle for reuse is the principle of *abstraction.* Abstraction consists of extracting the inherent, essential aspects of an artifact, while hiding its irrelevant or incidental properties. It fosters reuse by providing a class of artifacts that can be instantiated or customized to produce several different artifact instances meeting different requirements. The abstraction principle can also be used to automate the construction of artifacts that would normally require a creative process. Examples of successful application of this principle can be found in commercially available expert system shells, application generators, and more recently in algorithm synthesis [5] in the KIDS system which contains abstractions of several different classes of algorithms in the form of algorithm theories which can be (semi-) automatically instantiated to synthesize specialized algorithms for different problem instances.

The goal of the KASE project is to support software designers in designing software systems more efficiently and more reliably. We believe that designing software systems is a creative and ill-understood process that is performed by a small group of designers. However, the process and the final design is typically not well documented leading to poor reuse and expensive maintenance. Our approach to this problem is based on abstracting both the software system design and the design process. In brief, our approach consists of (1) identifying useful classes of software systems and the problems they solve, (2) abstracting the design of the system as a generic architecture for that class of problems, (3) abstracting the design process in the form of rules and procedures for customizing the generic architecture based on specific problem descriptions, (4) providing a computational environment that enables designers to construct specific systems semi-automatically by customizing the generic architecture.

A guiding theme in our research is to provide a set of software tools that support the way humans design. The objective is to produce a mixed-initiative system in which the design task is divided between a human designer and the system. Typically, KASE provides design alternatives and default suggestions for architectural parameters, explanations for its suggestions, dependency maintenance between different design decisions, and consistency checking. The human designers determines the order in which the various design actions are initiated, the final choice for each design decision

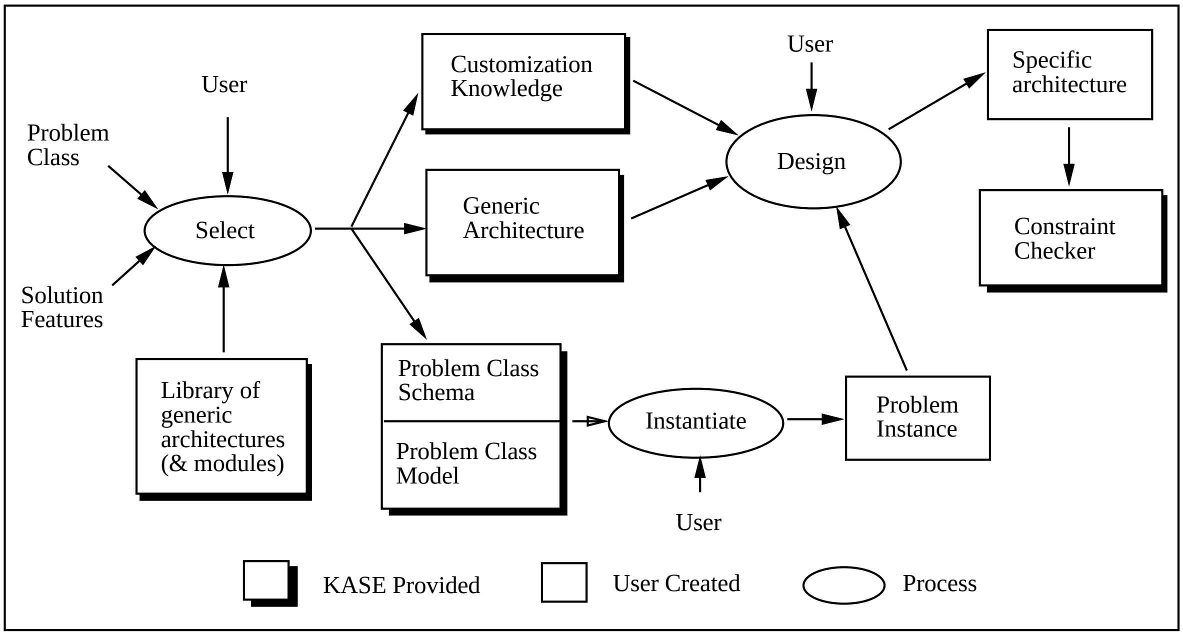

Figure 1: Overview of the KASE system

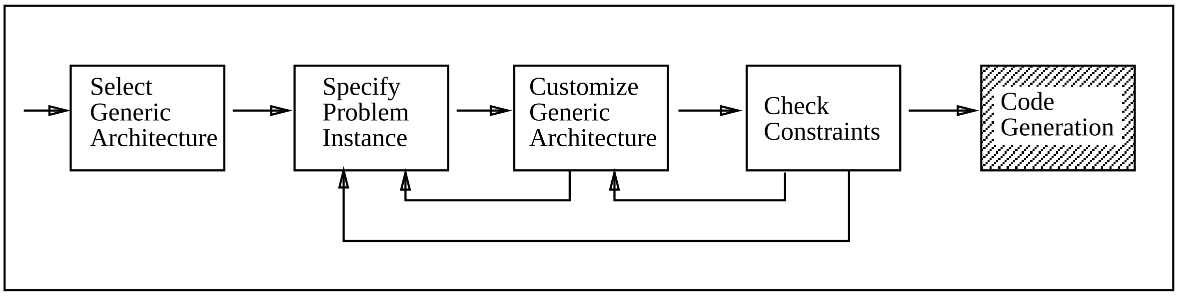

Figure 2: Process model for synthesizing software systems using KASE.

(which may or may not be based on the suggestions offered by KASE), and steps to be taken when a design inconsistency is reported. In essence, KASE acts as an assistant that delivers the relevant knowledge in the appropriate context to a designer so that he or she can make informed decisions and helps the designer in detecting errors early in the design process.

Figure 1 shows an overview of the KASE system. The shadowed boxes represent knowledge components that are part of KASE. Figure 2 illustrates the process of synthesizing systems using KASE.

A designer initiates the design process by first selecting a *generic architecture* from a library based on the *problem class* for his particular problem and the desired *solution features*. Each problem class has an associated specification in the form of a *problem-class schema* as well as a *problem-class model*. An individual *problem instance* is specified by instantiating the problem class schema; the problem class model contains the vocabulary of terms that help in the instantiation. Associated with pairs of generic architectures and problem classes is *customization knowledge* which contains knowledge for customizing the generic architecture and is the basis for KASE's intelligent support. Finally, the *constraint checker* is used to check for the consistency of the design with respect to certain architecture-specific constraints.

Our approach may be characterized as a semi-formal approach. It is not completely formal where the semantics of a problem specification and architectural descriptions are contained entirely within a set of mathematical equations. Nor is it completely informal where the name of a symbol carries all the information for a human is in, e.g. systems like IBIS [6]. Our approach does use keywords and assumes the availability of an accepted domain-specific ontology which is not formally defined. However, there are explicitly represented constraints and rules that provide some semantics to the symbols. We were motivated in adopting this approach because we wanted to create a practical system that could be used by software designers not well-versed in formal, mathematical notation; and, at the same time we wanted a representation that is interpretable by a machine so that it could provide intelligent assistance to designers.

3 Comparison

KASE shares the general goal of the Domain Specific Software Architecture (DSSA) project sponsored by DARPA which is to demonstrate the benefits of synthesizing software systems by reusing domain specific architectures. We have demonstrated the applicability of our approach in two different domains [2, 3]: a tracking domain and a graphical interface domain. In each of these domains a common generic architecture and customization knowledge was used to generate specific designs for different problem instances.

The LEAP project at Lockheed [7] and the ROSE-2 system [8] represent two other works that aim to synthesize systems by reusing high-level design abstractions. In LEAP there is a greater emphasis on code generation from a high level specification whereas in KASE and ROSE-2 there is a greater emphasis on high-level design issues and their representation.

Other related work on software architectures includes the work by researchers at Carnegie Mellon University (e.g. [9, 10]). Their work includes the identification of commonly used architectural paradigms, the relationship between various architectural paradigms and problem classes, and analysis of trade-offs involved in choosing one design over another /citeSha91. Lane [10] describes a user-interface software architecture and design rules for building specific user interface systems. The approach consists of creating a space of design alternatives and formulating rules that indicate good and bad design choices based on problem requirements - which is quite similar to the approach in KASE. However, it seems that there is less support for semi-automated or automated conversion of the resultant design into code.

There is some similarity between our approach of synthesizing software systems and parameterized programming. In parameterized programming, a generic program represents a parameterized algebraic theory which can be instantiated by using a view (theory morphism) that binds actual and formal parameters. from this perspective, KASE can be thought of as providing a library of highly parameterized programs (the generic system design) as well as tools to assist a user in instantiating them based on the requirements of a particular application.

References

[1] S. Bhansali and M. T. Harandi, "Synthesis of unix programs using derivational analogy," *Machine Learning*, vol. 10, no. 2, pp. 7–55, 1993.

[2] S. Bhansali and H. P. Nii, "Software design by reusing architectures," in *Proceedings of the 7th Annual Knowledge-Based Software Engineering Conference*, (McLean, Virginia), 1992.

[3] S. Bhansali, "Architecture-driven reuse of code in KASE," in *Proceedings of the 5th International Conference on Software Engineering and Knowledge Engineering*, (San Francisco Bay, CA), 1993.

[4] P. Garg and S. Bhansali, "Process programming by hindsight," in *Proceedings of the 14th International conference on software engineering*, (Melbourne, Australia), May 1992.

[5] D. R. Smith, "KIDS: a semiautomatic program development system," *IEEE Transactions on Software Engineering*, vol. 16, pp. 1024–1043, September 1990.

[6] J. Conklin and M. Begeman, "gibis: a tool for all reasons," *Journal of the American Society for Information Science*, vol. 40, pp. 200–213, 1989.

[7] H. Graves, "Lockheed environment for automatic programming," in *6th Annual Knowledge-Based Software Engineering Conference*, (Syracus, NY), pp. 78–89, 1991.

[8] M. Lubars, "The ROSE-2 Strategies for Supporting High-Level Software Design Reuse," Tech. Rep. STP-303-90, Microelectronics and Computer Technology Corporation, 1990.

[9] M. Shaw, "Heterogenous design idioms for software architecture," in *Fifth International Workshop on Software Specification and Design*, pp. 143–146, 1991.

[10] T. G. Lane, "A design space and design rules for user interface software architectures," Tech. Rep. CMU-CS-90-176, Carnegie Mellon University, 1990.

4 Biography

Sanjay Bhansali is a research scientist in the computer science department at Stanford University. He has been co-leading the Knowledge Assisted Software Engineering (KASE) project with H. Penny Nii at the Knowledge Systems Laboratory at Stanford for the past 2 years. The goal of the KASE project is to build a knowledge-based software design tool that can can act as an assistant to a human designer. Sanjay obtained his PhD in Computer Science at the University of Illinois, Urbana-Champaign in 1991 and his Bachelor of Technology in Computer Science from the Indian Institute of Technology, Delhi in 1986.