Formalization and Visualization of Domain-Specific Software Architectures

Paul D. Bailor, David R. Luginbuhl, and John S. Robinson

Department of Electrical and Computer Engineering
Air Force Institute of Technology
Wright-Patterson Air Force Base, Ohio 45433
pbailor@galaxy.afit.af.mil
(513) 255-3708

1 INTRODUCTION

This paper describes a domain-specific software design system based on the concepts of software architectures engineering [Lee and others, 1991] and domain-specific models and languages [Prieto-Díaz and Arango, 1991]. In this system, software architectures are used as high level abstractions to formulate a domain-specific software design. The software architecture serves as a framework for composing architectural fragments (e.g., domain objects, system components, and hardware interfaces) that make up the knowledge (or model) base for solving a problem in a particular application area [Lee and others, 1991]. A corresponding software design is generated by analyzing and describing a system in the context of the software architecture [Lee and others, 1991]. While the software architecture serves as the framework for the design, this concept is insufficient by itself for supplying the additional details required for a specific design. Additional domain knowledge is still needed to instantiate components of the architecture and develop optimized algorithms for the problem domain. One possible way to obtain the additional details is through the use of domain-specific languages. Thus, the general concept of a software architecture and the specific design details provided by domain-specific languages are combined to create what can be termed a domain-specific software architecture (DSSA).

2 DESCRIPTION OF DOMAIN SPECIFIC SOFTWARE DESIGN SYSTEM

The overall goal of our research is to prototype the technology required to formally specify, design, and develop an Ada application system using the DSSA approach described above. A key part of this effort is the creation and use of formal, domain-specific languages to generate software architectures whose architectural fragments and associated composition rules are maintained in a formal knowledge base of objects. These languages allow definition of the objects making up the components of the DSSA in terms of the components' structural and behavioral properties. Additionally, the domain-specific languages are used to compose the defined objects into a corresponding software design and resulting Ada implementation. In fact, the production rules of the grammar for the domain-specific language can serve as the basis for system composition. This is consistent with the approach suggested by Batory for composing hierarchical software systems with reusable components [Batory and O'Malley, 1992].

From a user interface perspective, software engineers use the domain-specific languages to define object classes and object composition rules to be placed into the knowledge base of objects. Alternately, application specialists (system end-users) use the languages to introduce new object instances and to compose object instances that currently exist in the knowledge base. Within the knowledge base, the objects and composition rules are maintained as executable, formal specifications providing the software engineer and application specialist with the ability to rapidly prototype and validate desired system behaviors without having to build Ada components first.

In addition to the domain-specific languages, some type of object base language is required to formalize the architecture and corresponding design representation. Such an object base language would also provide the ability to analyze and manipulate the objects defined and composed by the domain-specific language. This object base definition and manipulation language is used for the continued development of the domain-specific design and corresponding Ada software components. That is, the manipulation language is used to

formally manipulate the architectural objects to obtain a corresponding Ada design representation and Ada implementation of that object. Also, the object base manipulation language is the key to developing an application system that is composed of existing and validated Ada software components.

Thus, a formal framework for defining software architectures and domain specific languages would have to consist of the components listed below. The relationship or general configuration of the components is shown in Figure 1.

1. A *formalized object base* that serves two functions:
   (a) Formal specification of the concept of a software architecture consisting of a set of general abstractions associated with software architectures and a mathematical model of these abstractions.
   (b) Formal specifications of architectural fragments and instances of these fragments developed through the use of domain-specific languages as well as the object base manipulation language.
2. Formal specifications of *domain-specific languages* for describing and manipulating objects in the DSSA.
3. An *Ada development capability* that uses the formal specification of the architectural components to generate Ada components and allows for the composition of existing and validated Ada components into an application system.
4. A sophisticated *user interface* for both the application specialist and the software engineer. Note that visualization capabilities for both the domain-specific language constructs and the object base are highly desirable components of the user interface.

For this research effort, a prototype implementation of the technology will be done using the Software Refinery^{TM} Environment [Systems, 1990] that consists of the following components.

1. The Refine wide-spectrum language.
2. The Refine formal object base that is analyzed and manipulated via the Refine language.
3. The Dialect tool that allows for the definition of formal languages whose syntactical structures are directly mapped to objects in the object base. Note that this mapping is done in such a way that an abstract syntax tree relationship is maintained between the language components and corresponding objects in the object base. This relationship provides a significant advantage for language transformation purposes.
4. The Intervista tool that provides an X-windows based capability for graphical interaction with the object base.

Figure 2 graphically depicts the Refine framework. It provides an ideal platform to prototype the proposed DSSA technology. The Dialect and Intervista Tools are used for the User Interface aspect as they provide the means to define domain-specific languages, map domain language structures to a formal object base, and visualize both the domain language structures and the software architecture. The Refine language provides the means to manipulate objects in the object base for performing operations such as transforming the formalized objects into Ada components, analyzing the object's behavior for validation/verification purposes, and composing sets of objects into a higher level application system. An important advantage of the Refine framework is that it reduces tool development time to zero. Thus, it allows the research to focus on the development of the new technology immediately.

3 Research Issues

The domain-specific software design system described above has several important research issues associated with it that we are currently attempting to address.

1. What are the abstractions associated with the concept of a software architecture, and how can we formally model these abstractions?
2. What is the feasibility of developing the required domain-specific languages? There are a number of relevant Air Force application domains that have already been analyzed and at least partially structured using the concepts of a software architecture; for example, the electronic combat domain of The Joint Modeling and Simulation System (J-MASS) [ASD/RWWW, 1990, ASD/RWWW, 1991], the C^{3}I domain [Plinta and Lee, 1989], and the radar tracking domain [Jensen and Ogata, 1991]. Additionally, the DARPA Domain-Specific Software Architecture project is funding research in an attempt to define software architectures in four application areas. All of these could serve as candidates for development of domain-specific languages; however, we must first address three important sub-issues:
   (a) How difficult is it in general to encode the domain-specific knowledge required to compose objects in the domain into the production rules of a grammar? Alternatively, how difficult is it to develop and formalize the domain-specific knowledge required for this and place it into a knowledge base of composition rules?

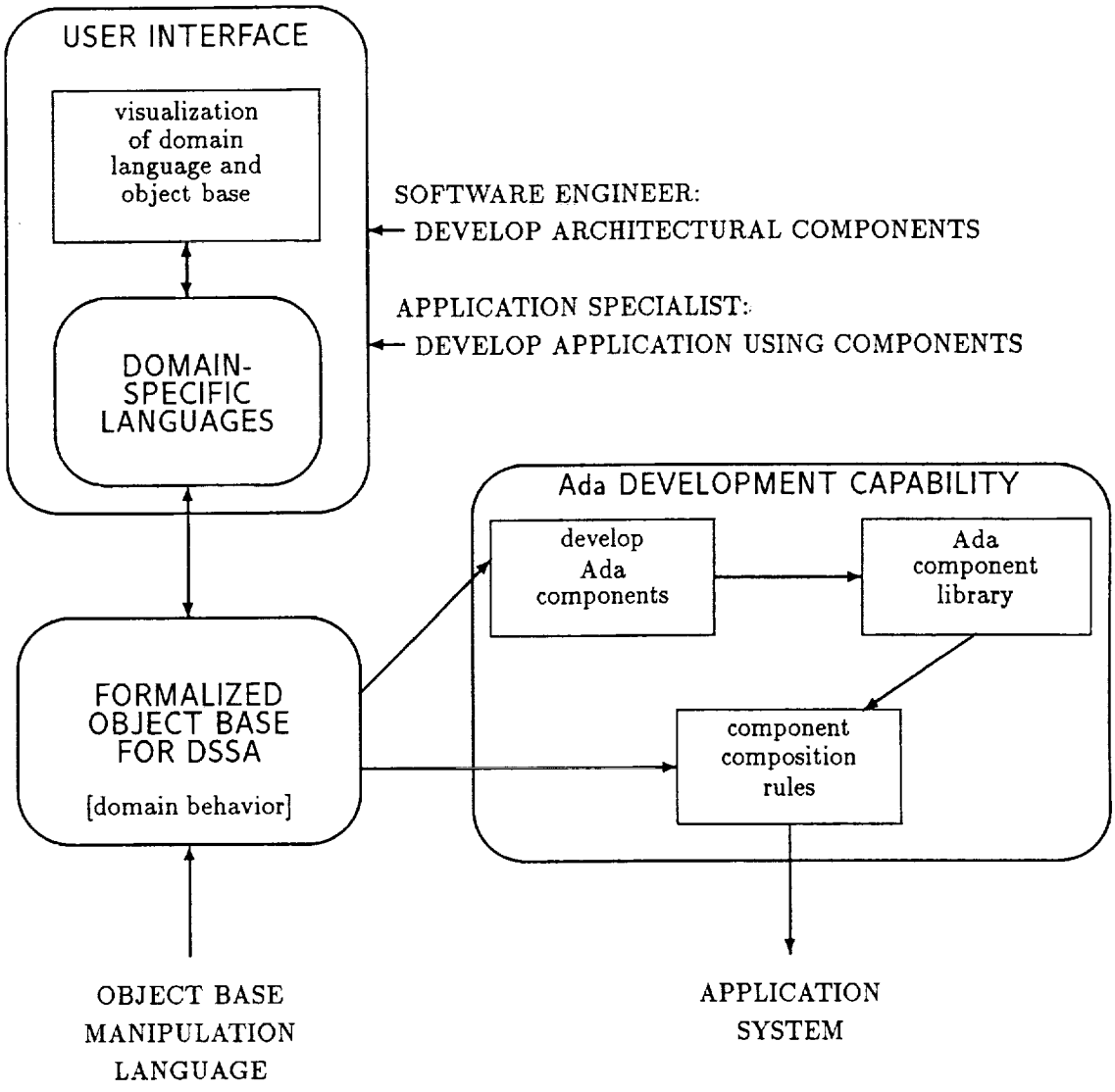

Figure 1: General Configuration for Formalizing a DSSA

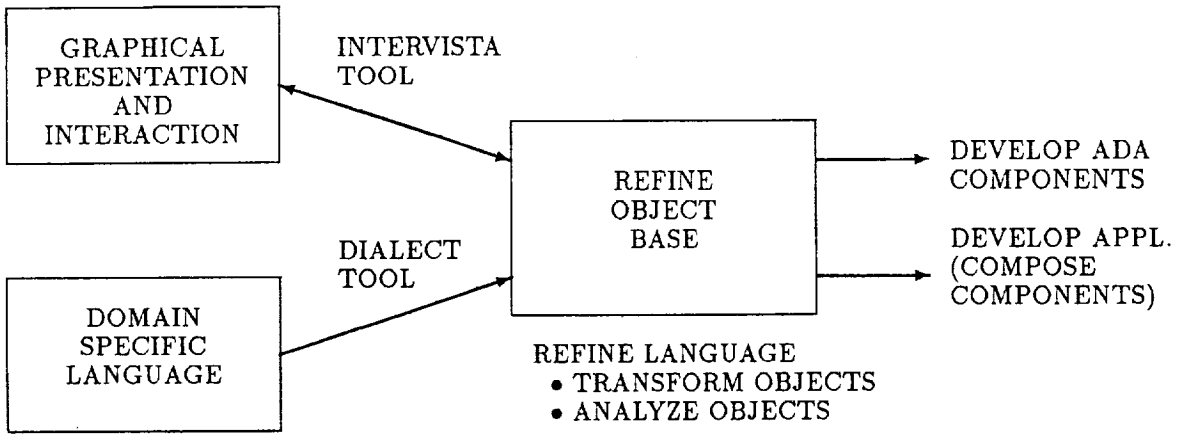

Figure 2: Refine System Framework

(b) Can the same domain-specific language be used by both the software engineer and the application specialist?

(c) Is there a core of language constructs that are common to all domain languages?

3. What is the feasibility of using a knowledge-based transformation system to develop a highly visual interface to the domain-specific software design system that is useful to both the software engineer and the application specialist?

We have focused our short-term research objectives towards addressing the above issues first. Specifically, the research objectives for the first two years of this effort are to:

1. Define the abstractions associated with the concept of software architectures and develop a mathematical model of these abstractions. The Refine system will be used to prototype and analyze formal models of software architectures.

2. Analyze a part of the electronic combat domain and develop the required formal domain language(s).

3. Use the Software Refinery Environment to build a working prototype of our DSSA system that includes the ability to build a formal object base of architecture fragments and to apply composition rules to the fragments to construct domain-specific software designs in the form of a DSSA.

4. Develop visualizations of both the formal domain language(s) and the object base.

In the following years, we expect the major concentration to be on using the formalized object base as a basis for developing the corresponding Ada components and as a basis for composing an application system within a domain. Additionally, methods and tools for scaling the technology up for large scale applications will be investigated.

4 SUMMARY

We feel the proposed research can have a significant impact on the methodologies for implementing the concepts of software architectures and domain-specific software design, especially in the areas of formalizing software architectures and formalizing the application of domain-specific languages to software specification and design. Additionally, this research should provide much insight into the process of object-oriented development of validated and reusable Ada components that can be quickly and validly composed into an application system for a particular domain.

References

[ASD/RWWW, 1990] ASD/RWWW. Joint Modeling and Simulation System (J-MASS): System Concept Document. Technical report, CROSSBOW-S Architecture Technical Working Group, December 1990.

[ASD/RWWW, 1991] ASD/RWWW. Software Structural Model Design Methodology. Technical report, Architecture Technical Working Group, June 1991.

[Batory and O'Malley, 1992] Don Batory and Sean O'Malley. The Design and Implementation of Hierarchical Software Systems With Reusable Components. Technical Report TR-91-22, Department of Computer Sciences, University of Texas at Austin, Austin, Texas, January 1992.

[Jensen and Ogata, 1991] Paul S. Jensen and Lori Ogata. Final Report for Automatic Programming Technologies for Avionics Software (APTAS). Technical Report LMSC-P000001, Lockheed Software Technology Center, Palo Alto, California, July 1991.

[Lee and others, 1991] Kenneth J. Lee et al. Model-Based Software Development (Draft). Special Report CMU/SEI-92-SR-00, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, December 30 1991.

[Plinta and Lee, 1989] Charles Plinta and Kenneth Lee. A Model Solution for the C^3I Domain. In *Tri-Ada Conference*, pages 56–67. New York: ACM Press, 1989.

[Prieto-Díaz and Arango, 1991] Prieto-Díaz and Arango. *Domain Analysis and Software Systems Modeling.* IEEE Computer Society Press : California, 1991.

[Systems, 1990] Reasoning Systems. Refine User's Guide. Reasoning Systems, Inc., 1990.