ARPA Domain Modeling Workshop, George Mason University, September 1994

Products of Domain Models[1]

Don Batory
Department of Computer Sciences
The University of Texas
Austin, Texas 78712
batory@cs.utexas.edu

Abstract

We argue that domain models should produce four basic products: identification of reusable software components, definition of software architectures that explain how components can be composed, a demonstration of architecture scalability, and a direct relationship of these results to software generation of target systems.

1 Introduction

Conventional application modeling is aimed at producing software designs for one-of-a-kind applications. Domain modeling, in contrast, takes a broader view to address designs for families of related applications. The issues or products that distinguish domain modeling form application modeling include:

- reusable software components
- software architectures
- scalability
- software system generators

Although domain modeling can be justified for many reasons, we feel the ultimate benefit of domain modeling is the development of tools that eliminate the mundane tasks of application programming within a mature domain. That is, the automatic construction of software seems to be an inevitable end-product of the evolution for a well-understood domain; we see domain modeling as the means to achieve this end.

2 Products of Domain Models

The identification of *reusable software components* is a critical product of domain modeling. The key to their identification is recognizing fundamental abstractions of the domain. These recurring abstractions appear in most systems of a family; by standardizing their programming interfaces, components that implement these abstractions are plug-compatible, interchangeable, and interoperable. Plug-compatibility, interchangeability, and interoperability increases likelihood that components will be reused.

The second product of domain modeling is *software architectures* for a domain. A software architecture basically is a blue-print or template for defining how components fit together to form systems. Software architectures express patterns of composition that recur within a domain, it explains various means by

1. This research was sponsored, in part, by the U.S. Department of Defense Advanced Research Projects Agency in cooperation with the U.S. Air Force Wright Laboratory Avionics Directorate under contract F33615-91C-1788.

which compositions (or component communications) can physically occur, and it suggests guidelines for selecting compositions for a given task [Gar92].

Scalability is the third product. It addresses the need for architectures and their components to evolve. The ability to add new features to systems is the driving force for modeling families of systems; the need for new features will always be present. Domain models must clearly demonstrate the capability of scaling to ever larger families of systems without requiring significant revisions [Big94].

Domain models should be detailed enough to be used as blue-prints for *software system generators*; i.e., tools that can produce a target system from a domain directly from specifications. Specifications may express high-level constraints to be satisfied, thereby entailing a degree of automatic programming on the part of the generator to "fill in the gaps", or it could rely exclusively on the domain analyst for specific selections and compositions of components [Pou94]. Software system generators are among the most visible and concrete products that can come from domain modeling; a generator is often the centerpiece for other analysis tools that are critical to applications development [Bat93].

3 Recap

We have an admittedly narrow focus on what should be the products of domain modeling; we concur that a broader vision should be adopted by the software engineering community. However, we feel our focus is justified as a consequence of years of work in the areas of reuse, software architectures, and software generators [Bat92-94]. For us to have demonstrated anything less than software system generation on our projects would not have been sufficient to convince sponsors of the importance and viability of our concepts of domain modeling. We believe that the demand for such concrete results is universal, and the products that we have identified go a long way for justifying the effort and commitment needed for domain modeling.

4 References

[Bat92] D. Batory and S. O'Malley, "The Design and Implementation of Hierarchical Software Systems with Reusable Components", *ACM Transactions on Software Engineering and Methodology*, October 1992.

[Bat93] D. Batory, L. Coglianese, M. Goodwin, and S. Shafer. "Creating Reference Architectures: An Example from Avionics", ADAGE Tech. Rep. UT-93-06, 1993.

[Bat94] D. Batory, V. Singhal, J. Thomas, S. Dasari, B. Geraci, and M. Sirkin, "The GenVoca Model of Software-System Generators", to appear in *IEEE Software*, September 1994.

[Big93] T. Biggerstaff, "An Assessment and Analysis of Software Reuse", in *Advances in Computers*, Vol. 34, Academic Press, 1992.

[Big94] T. Biggerstaff, "The Library Scaling Problem and the Limits of Concrete Component Reuse", 3rd Conference on Software Reuse, Rio de Janerio, November 1994.

[Gar94] D. Garlan and M. Shaw, "An Introduction to Software Architecture", in *Adances in Software Engineering and Knowledge Engineering*, Volume I, World Scientific Publishing Company, 1993.

[Pou94] J. Poulin and W. Tracz, "WISR'93 Reuse Workshop Summary", *ACM Software Engineering Notes*, Vol. 19 #1, January 1994.