

Generative Techniques for Product Lines — An ICSE 2001 Workshop Report

Greg Butler
Department of Computer Science, Concordia University
Montreal, Canada
e-mail: gregb@cs.concordia.ca

Abstract

A software product line leverages the knowledge of one or more domains in order to achieve short time-to-market, cost savings, and high quality software. The highest level of reuse comes by using domain-specific languages or visual builders to describe a member of the product line, and to generate the member from the description. Generative techniques can help us to capture the configuration knowledge for a product line and use it to generate concrete family members. This workshop focuses on technical issues of product lines, rather than economic issues.

Introduction

Software product lines have demonstrated high levels of software reuse and achieved short time-to-market, cost savings, and high quality software [2]. As our understanding of a particular domain matures, it is likely that the product line will evolve to be an application generator [1]. Even at an early stage in the maturity of a software product line, there is ample scope to apply generative techniques [3]. The highest level of reuse comes by using domain-specific languages (DSLs) or visual builders to describe a member of the product line, and to generate the member from the description.

A very successful workshop at the First Software Product Line Conference [4] in Denver on August 29, 2000 identified some conclusions (or conjectures) about the state-of-the-art in generative techniques:

- Limit the scope of domain so it is small, simple, specialized.
- There are three kinds of jargons for DSLs: procedural, data, mark-up;
- DSLs can generate multiple artifacts.
- DSLs should have multiple views of the domain.
- Scalability is important: we need composition of domains.
- Domains evolve: so practice maintenance, and configuration management.
- There is a need for better tools: tools for feature model, extensible IDE supporting interaction/feedback in domain terminology (generation, error messages, debugging, profiling, optimization).

- The skill set for DSL developer is not same as skill set for domain expert nor software developer.

That workshop also identified many open issues:

1. How do you validate a product line, for example, the domain specification, DSLs, generators, DSL specification of a product, and the product itself?
2. Are there methodologies for partitioning a domain into smaller/simple/more specialized subdomains?
3. How do you compose domains?
4. How to manage the evolution of domains and DSLs?
5. How do we share domain knowledge across organizations within an industry?
6. How do you do domain knowledge acquisition and engineering?

The Workshop on Generative Techniques for Product Lines at ICSE 2001 [5] built on the above progress. The participants were

- Greg Butler, Concordia University, Montreal: facilitator;
- Michel Barbeau, Carleton University, Ottawa, Canada: telecom software;
- Ira Baxter, Semantic Designs, Inc, Austin, Texas: domain modeling and transformations;
- Francis Bordeleau, Carleton University, Ottawa, Canada: generative techniques in real-time system design;
- Markku Hakala, Tampere University of Technology, Finland: the FRED environment for architecture-oriented programming;
- Juha Hautmaki, Tampere University of Technology, Finland: the FRED environment for architecture-oriented programming;
- John MacGregor, Robert Bosch GmbH, Germany: product lines for automotive software;
- Dirk Muthig, Fraunhofer Institute for Experimental Software Engineering, Germany: product line methodology;

- Markus Pizka, Technische Universität München, Germany: applications of generative programming to operating systems;
- Andreas Rausch, Technische Universität München, Germany: *Zen*, a code generator based on XML and code templates; and
- Kurt Stirewalt, Michigan State University, USA: generative techniques and code generation.

Activities

The workshop was small and informal. There was an opportunity for everyone to present their work at leisure, and there was an extended demonstration of the FRED environment. The presentations and demonstration invoked many questions and much discussion.

The position papers are on the workshop web page www.cs.concordia.ca/~faculty/gregb/icse-workshop/ so here we will focus on the discussion arising from the presentations and demonstrations.

Andreas Rausch presented *Zen*, a generator using XML for the DSL and code fragments defining the generic target source code. He noted that there were issues of scaleability, and of maintenance of the XML and code fragments. The discussion topics covered (1) whether non-functional constraints could be included in the DSL and whether more “intelligent” generators could make use of the non-functional constraints during code generation; (2) if one adopted an aspect-oriented approach, where would one weave in a description of *solutions* to the non-functional constraints; and (3) a general criticism of tools for product lines, since these are distinctly different from tools for application development.

Markus Pizka presented early work on *MoDis*, a distributed operating system product line based on the Space Time Abstraction-level system model of resources, bindings, tasks, and structuring. There as a view of an OS as a product line, but also a view of an OS as a generator (job descriptions to machine code). Many sets of DSLs are involved. Some issues raised were how to treat mixed conflicting requirements, especially performance versus modularity, and whether that requires that one expose parts of the system design and operation. The OS community still believes that a monolithic OS is required for performance, and has not absorbed the lessons of Don Batory’s work on Genvoca [1] which solves the dilemma of modularity versus performance. The discussion dealt with (1) the challenge of handling multiple DSLs within a system, the systematic derivation of DSLs, and capturing and resolving interdependencies between DSLs; and (2) the essence of the product line definition: what criteria are there for inputs, and what criteria are there for outputs of a product line generator?

Markku Hakala demonstrated *FRED*, an environment that supports the development of generic architectures by the selection of patterns. FRED has generic text descriptions of

how to apply each pattern, role based instantiation of patterns, and pattern descriptions as graph grammars. Patterns could be associated with code templates, and FRED maintained the actual binding between the architecture and the generated code as part of the description of an application architecture. The question of how to evolve or refine the generic architecture to include abstract interface models with contracts or to include behavioural models using MSCs was raised. The issue of describing pattern families and the dependencies between patterns also arose.

John MacGregor listed several projects involving product lines that he had worked on, noting limitations with feature modeling and aspects that could possibly be resolved with knowledge representation techniques for ontology. He also raised the issue of validation of domain models (see Issue 1 above).

Francis Bordeleau presented their approach to generating code for real-time systems using models of scenarios, interaction diagrams, ROOM role model, and system model. He proposed *integration patterns* to capture proven solutions for composition of modules of real-time systems. A major issue was that of standards: there being no standard C, nor MSC standard, and there will likely be variants of UML semantics. This complicates the adoption of such standards for input to generators.

Conclusions

One part of the discussion focused on the state-of-the-art in generative techniques, and which techniques were ready to be introduced to an industrial setting. Our summary of best practice is

- template metaprogramming (see [3], for example);
- the FAST approach to product line development at Lucent [6];
- Don Batory’s line of research on Genvoca that produced Predator, P++, mixin layers, and the Jakarta Java tools (see his web page www.cs.utexas.edu/users/dsb/);
- Statecharts and the code generators from I-Logix, ObjectTime, and Rational;
- transformation/refinements techniques, such as incorporated in XSL transformations of XML documents, and more generally by Semantic Designs, Inc;
- refactoring of source code where tools exist for Smalltalk and Java code;
- tools for compiler generation or macro generation as found in Unix; and
- database DDLs (data definition languages) and 4GL report generators (which may, however, soon be subsumed by XML technology).

Another part of the discussion focused on ideal, desirable, or necessary properties of a generator. While we did not classify the properties, we did develop a list of properties:

- A generator transforms a set of models to a set of models.
- In the context of a product line, one input to the generator is a selection of features.
- The generator should provide guidance in the selection of features, or alternatively, present the selection as a list of design decisions to be made by the application designer.
- Generators should be able to deal with incomplete input, such as partial models or incomplete feature sets, and offer the possibility of either the generator making default choices for the missing parts, or of the generator interacting with the application designer to elicit the missing information.
- A generator should provide a trace of the derivation of the output models.
- A generator should allow incremental operation, whereby a change of selected features implies a reuse of the derivation.
- A generator should be “open”, in that the transformation rules are visible, modifiable, and extensible by an expert “generator engineer”.

The workshop emphasized the importance of several open issues, since each arose repeatedly throughout the day. These were Issues 2 and 3 on scalability; Issue 4 on the evolution of domains and models; and a new issue on the need for meta-level generator generators.

Organizers

Greg Butler is a Professor of Computer Science, Concordia University, Montreal. He works on frameworks, investigating methodologies for framework development and evolution.

Don Batory holds the David Bruton Centennial Professorship at The University of Texas at Austin. He has given numerous tutorials on Product-Line Architectures, Generators, and Reuse, and is an industry-consultant on product-line architectures.

Krzysztof Czarnecki is a researcher and consultant with the Software Technology Lab at DaimlerChrysler Research in Ulm, where he has been working on Generative Programming and its industrial application for over four years.

Ulrich Eisenecker is a professor of computer science at the University of Applied Sciences, Kaiserslautern. His work focuses on generative programming and object technology. He is also the editor of KOMPONENTEN-Forum, which is a permanent part of OBJEKTspektrum, a SIGS publication on object and component technology in Germany.

References

- [1] D. Batory and S. O’Malley. The design and implementation of hierarchical software systems with reusable components. *ACM Trans. on Software Engineering and Methodology*, 1(4):355–398, 1992.
- [2] G. Butler. Quality and reuse in industrial software engineering. In *Proceedings of Asia-Pacific Software Engineering Conference and International Computer Science Conference*, pages 3–12. IEEE Computer Society Press, 1997.
- [3] K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison Wesley, 2000.
- [4] P. Donahue, editor. *Software Product Lines: Experience and Research Directions*. Kluwer Academic, Boston, 2000.
- [5] *Proceedings of the 23rd International Conference on Software Engineering*. IEEE Computer Society, Los Alamitos, CA, 2001.
- [6] D. M. Weiss and C. T. R. Lai. *Software Product-Line Engineering*. Addison-Wesley, 1999.