

# Model-Driven Software Product Lines

Krzysztof Czarnecki, Michał Antkiewicz,  
Chang Hwan Peter Kim, Sean Lau, Krzysztof Pietroszek  
University of Waterloo  
200 University Ave. West  
Waterloo, ON N2L 3G1, Canada  
{kczarnec,mantkiew,chpkim,sqlau,kmpietro}@swen.uwaterloo.ca

## ABSTRACT

Model-driven software product lines combine the abstraction capability of Model Driven Software Development (MDS) and the variability management capability of Software Product Line Engineering (SPLE). This short contribution motivates the idea of model-driven software product lines and briefly explains the concepts underlying *feature-based model templates*, which is a particular technique for modeling software product lines.

## Categories and Subject Descriptors

D.2.1 [Software Engineering]: Requirements / Specifications—*Tools*; D.2.2 [Software Engineering]: Design Tools and Techniques—*Computer-aided software engineering (CASE)*; D.2.4 [Software Engineering]: Software/Program Verification; D.2.13 [Software Engineering]: Reusable Software—*Domain engineering, Reuse models*

## General Terms

Design, Documentation, Verification

## Keywords

Domain analysis, feature modeling, model-driven software development, product configuration, software-product lines, software reuse, variability modeling and management

## 1. INTRODUCTION

Software product line engineering and model-driven software development are two recent trends that have been drawing increased attention from the software development community. *Software product line engineering* (SPLE) [10, 1] optimizes the development of individual systems within an application domain by leveraging their common characteristics and managing their differences in a systematic way. In SPLE, individual systems can be built rapidly from reusable assets, such as a set of components and/or a common platform.

*Model-driven software development* (MDS) aims at capturing every important aspect of a software system through appropriate models. Compared to implementation code, models capture the intentions of the stakeholders more directly, are freer from accidental implementation details, and

are more amenable to analysis. In MDS, models are not just auxiliary documentation artifacts; rather, they are source artifacts and can be used for automated analysis and/or code generation.

Generative software development [2] and related approaches, such as Software Factories [6], have been propagating the integration of software product lines and model-driven software development; also, an entire workshop has been recently dedicated to this topic [9]. At the root of this development lies the recognition that SPLE and MDS are not only complementary, but their integration bears the potential for significant synergies. While MDS can help us represent different aspects of a product line more abstractly, SPLE provides a well-defined application scope, which puts the development and selection of appropriate modeling languages on a sound basis. Furthermore, the automated analysis and code generation afforded by precise models can help us automate the creation of product line members.

A particular technique for model-driven development of product lines that was proposed in our research group is *feature-based model templates* [3]. In a nutshell, a model template can represent a set of model variants in a superimposed form within a single artifact. The represented variants could be behavioral, structural, or non-functional models. Furthermore, a feature-based model template also contains a feature model which concisely represents the available choices within the set of variants. Thus, feature-based model templates combine two ingredients: feature modeling and model templates, which we describe as next.

## 2. FEATURE MODELING

*Feature modeling* is a technique for representing the commonalities and the variabilities among a set of systems in concise, taxonomic form [7]. Features are prominent functional and/or nonfunctional characteristics of the systems. Feature models are hierarchies of features that describe different kinds of variability. For example, some features are mandatory and some are optional or alternative.

We have developed a particular form of feature modeling, which is referred to as *cardinality-based* [5]. In cardinality-based feature modeling, the number of possible selections from a group of features is specified by a cardinality interval. Furthermore, a feature can have a cardinality interval expressing whether the feature can be selected or cloned. Finally, features can have attributes for representing values such as numbers and strings. These additional modeling facilities make the feature models applicable to a wider range of configuration problems, such as embedded software [4].

Feature models are requirements-level artifacts. They are useful for scoping product lines, i.e., deciding which features should be supported by the product line and which feature should not. They also provide a vocabulary to concisely describe the members of a product line. In particular, we can use them to create feature configurations as input to an automated product derivation process.

The actual semantics of features can be captured by other models, such as structural, behavioral, or non-functional models expressed in an appropriate modeling language. This idea brings us to the concept of feature-based model templates.

### 3. FEATURE-BASED MODEL TEMPLATES

A feature-based model template consists of feature models and annotated models implementing the features. The annotations refer to the features in the feature model and can have the form of presence conditions, iteration directives, and/or meta-expressions. Presence conditions are similar to the `#ifdef` directive in the C preprocessor, but applied to model elements. Iteration directives allow iterative template expansion, and meta-expression can be used to compute model elements. Our approach works for any notation defined using the Meta-Object Facility (MOF) [8], such as UML, but can also be adapted for other metamodeling formalisms.

As already mentioned, a model template represents a set of requirements-, design-, or implementation-level model variants in a superimposed form. This way, the variants can be maintained as a single artifact, rather than individually. Furthermore, the annotations provide traceability between features and the model elements that realize them. Finally, we can produce a template instance for a given feature configuration automatically. An automated verification procedure ensures that no ill-structured template instances can be produced.

As a proof of concept, we have built a set of tools supporting our approach. The set contains (1) *fmp*, an Eclipse plug-in for feature modeling and configuration; (2) *fmp2rsm*, a plug-in that adds support for model templates to IBM Rational Software Modeler, an Eclipse-based UML2 modeling environment; and (3) *template verifier*, an extension to *fmp2rsm* that can verify for a given template that no ill-defined template instances can be produced from a valid feature configuration.

Our other contribution, which is located in the demonstration section of this volume, gives a more detailed description of these tools, followed by a brief discussion.

### 4. ABOUT THE AUTHORS

Krzysztof Czarnecki is an Assistant Professor in the Department of Department of Electrical & Computer Engineering (ECE), University of Waterloo, where he leads a research group on generative and model-based software engineering. Michał Antkiewicz and Krzysztof Pietroszek are PhD candidates in the Department of ECE, University of Waterloo. Sean Lau and Peter Kim are Master's degree candidates in the same department. Michał is responsible for *fmp* and *fmp2rsm*. Sean is responsible for validation of the approach. Peter is responsible for the constraint-based feature configuration and feature model synchronization. Finally, Krzysztof Pietroszek is responsible for the template

verifier.

### 5. REFERENCES

- [1] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, Boston, MA, 2001.
- [2] K. Czarnecki. Overview of Generative Software Development. In *Proceedings of Unconventional Programming Paradigms (UPP) 2004, 15-17 September, Mont Saint-Michel, France, Revised Papers*, volume 3566 of *LNCS*, pages 313–328. Springer-Verlag, 2004. <http://www.swen.uwaterloo.ca/~kczarnec/gsdoverview.pdf>.
- [3] K. Czarnecki and M. Antkiewicz. Mapping features to models: A template approach based on superimposed variants. In R. Glück and M. Lowry, editors, *GPCE 2005 - Generative Programming and Component Engineering. 4th International Conference, Tallinn, Estonia, Sept. 29 - Oct. 1, 2005, Proceedings*, volume 3676 of *LNCS*, pages 422–437. Springer, 2005.
- [4] K. Czarnecki, T. Bednash, P. Unger, and U. W. Eisenecker. Generative programming for embedded software: An industrial experience report. In D. Batory, C. Consel, and W. Taha, editors, *Proceedings of the ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering (GPCE'02), Pittsburgh, October 6-8, 2002*, volume 2487 of *LNCS*, pages 156–172, Heidelberg, Germany, 2002. Springer-Verlag.
- [5] K. Czarnecki, S. Helsen, and U. Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process Improvement and Practice*, 10(1):7–29, 2005.
- [6] J. Greenfield and K. Short. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley, Indianapolis, IN, 2004.
- [7] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Nov. 1990.
- [8] Object Management Group. *Meta-Object Facility*, 2002. <http://www.omg.org/technology/documents/formal/mof.htm>.
- [9] D. C. Schmidt, A. Nechypurenko, and E. Wuchner. MODELS'05 Workshop "MDD for Software Product-lines: Fact or Fiction?". <http://www.geocities.com/andreynech/MDDandProductLinesWorkshop.html>, 2005.
- [10] D. M. Weiss and C. T. R. Lai. *Software Product-Line Engineering: A Family-Based Software Development Process*. Addison-Wesley, Boston, MA, 1999.