Integrated Revision and Variation Control for Evolving Model-Driven Software Product Lines[1]

Felix Schwägerl[2] Bernhard Westfechtel[3]

Abstract: Software engineering projects are faced with abstraction, which is achieved by software models, historical evolution, which is addressed by revision control, and variability, which is managed with the help of software product line engineering. Addressing these phenomena by separate tools ignores obvious overlaps and therefore fails at exploiting synergies between revision and variation control for models. In this article, we present a conceptual framework for integrated revision and variation control of model-driven software projects and its implementation in the tool SuperMod.

Keywords: Model-Driven Software Engineering; Revision Control; Variation Control

1 Background

This work is concerned with the integration of three disciplines of software engineering: In *model-driven software engineering*, software systems are developed from high-level models that are analyzed, simulated, or transformed into source code. *Software product line engineering* denotes a systematic reuse process that supports the efficient development of instances of a product line from a set of shared artifacts. *Software configuration management* is the discipline of managing the evolution of complex software systems; in particular, it includes version control for software engineering artifacts such as models and source code.

To some extent, model-driven software engineering, software product line engineering, and software configuration management have been evolving independently: Model-driven software engineering focuses on models and model transformations without considering evolution in time and space. Software product line engineering addresses evolution in space only, primarily dealing with source code. Software configuration management is concerned with evolution in time, focusing on file-based artifacts. Partial integration of these disciplines has been studied. For example, model version control deals with historical evolution of models, and model-driven software product line engineering applies product line engineering to model- rather than code-based artifacts.

[1] This paper is an extended abstract of [SW19].

[2] Universität Bayreuth, Lehrstuhl für Angewandte Informatik 1, Universitätsstraße 30, 95440 Bayreuth, Germany felix.schwaegerl@uni-bayreuth.de

[3] Universität Bayreuth, Lehrstuhl für Angewandte Informatik 1, Universitätsstraße 30, 95440 Bayreuth, Germany bernhard.westfechtel@uni-bayreuth.de

doi:10.18420/SE2020_40

2 Conceptual Framework

In our work, we developed a *conceptual framework* for *managing evolving software product lines*. This framework is based on earlier work on a *uniform version model* for the domain of software configuration management [WMC01] and distinguishes between different layers of abstraction. The *base layer* provides generic support for versioning of artifacts of arbitrary types. Versioned artifacts are represented as *superimposition* of versioned elements. As in conditional compilation or annotative approaches to software product line engineering, versioned elements are decorated with *visibilities*, i.e., expressions that determine the versions in which an element is included. *Version rules* are used to constrain the combination of truth values that may be assigned to the variables occurring in visibilities.

On top of the base layer, user-level models for revision and variation control are realized. Here, familiar abstractions from software product line engineering and software configuration management are reused. *Revision control* (evolution in time) is supported by *revision graphs*. Each revision denotes an immutable snapshot of a software product line. For *variation control* (evolution in space), *feature models* are provided. Feature models are versioned in time; domain artifacts (models and source code) evolve in both time and space.

3 Tool Support

Based on the conceptual framework described above, we developed *SuperMod* (*Superimposition* of *Mod*els), a tool for managing evolving model-driven software product lines. The tool maintains *repositories* of versioned artifacts such as feature models, domain models, and source code. Furthermore, SuperMod offers operations such as *check-out* and *commit* to populate *workspaces* and record changes performed in the workspaces, respectively.

While the check-out/commit cycle was inspired by classical version control tools known from software configuration management, SuperMod differs considerably from version control tools with respect to variation control. On check-out, the user selects a revision first and then proceeds by selecting a variant. Thus, the workspace is populated with a unique product version. Both the feature model and the domain artifacts may be changed in the workspace. On commit, the set of variants is determined which are affected by the changes performed in the workspace. Altogether, this approach is called *dynamic filtered editing*.

Bibliography

[SW19] Schwägerl, Felix; Westfechtel, Bernhard: Integrated revision and variation control for evolving model-driven software product lines. Software and Systems Modeling, 18(6):3373–3420, December 2019.

[WMC01] Westfechtel, Bernhard; Munch, Bjørn P.; Conradi, Reidar: A Layered Architecture for Uniform Version Management. IEEE Transactions on Software Engineering, 27(12):1111–1133, 2001.