Towards a Model-Driven Product Line for SCM Systems

Thomas Buchmann
Lehrstuhl Angewandte Informatik 1
University of Bayreuth
thomas.buchmann@uni-bayreuth.de

Alexander Dotor
Lehrstuhl Angewandte Informatik 1
University of Bayreuth
alexander.dotor@uni-bayreuth.de

Abstract

Software configuration management (SCM) is the discipline of controlling the evolution of large and complex software systems. Many tools and systems for SCM have been developed which are based on a variety of different version models. Usually, the underlying version models have been hard-wired into the respective tool or system. In this paper we present MOD2-SCM, a PhD project which is dedicated to a model-driven approach to the development of SCM systems which makes the underlying version model explicit, reduces development effort by replacing coding with creating executable models, and supports reuse by providing a product line for SCM.

1 Introduction

Software configuration management (SCM) is the discipline of controlling the evolution of large and complex software systems. A wide variety of SCM tools and systems has been implemented, ranging from small tools such as RCS [25] over medium-sized systems such as CVS [28] or Subversion [6] to large-scale industrial systems such as Adele [12] and ClearCase [30].

Version control is a core function of any SCM system. Version control is based on *version models* which have been implemented in many research prototypes, open source products, and commercial systems [7]. An analysis of these systems shows that similar concepts such as revisions, variants, state- and change-based versioning appear over and over again. Unfortunately, version models are usually implicitly contained in implemented systems.

Thus, the SCM domain is characterized by a large number of systems with more or less similar features incorporating hard-wired version models which have been implemented with considerable effort. This observation has motivated us to set up a project dedicated to a **modular and model-driven product line for SCM** (abbreviated as **MOD2-SCM**) The overall goal of the MOD2-SCM project is to provide a novel way of constructing SCM systems. Our intent is to support the rapid construction of customized SCM systems with acceptable effort. As already mentioned, current SCM systems are usually monolithic systems which have been implemented with considerable effort, and can be adapted to the requirements of their users only to a limited extent. Instead, our vision is to compose a customized SCM system from a set of reusable modules in a flexible way.

The remainder of this paper is organized in ten sections. In section 2, a brief sketch of the technical problem is given, followed by a discussion of related work in section 3. Section 4 covers the research hypothesis while the proposed solution is sketched in section 5. The expected contributions are listed in in section 6. Section 7 gives an overview about the current state or our work. We conclude the paper by giving an overview about methods used for our research in section 8 and we present a plan for evaluation of our system in section 9. Finally, the conclusion follows in section 10.

2 Technical problem

The development of a product line for SCM systems is a complex process covering several areas of software engineering, e.g. requirements & domain engineering, as well as design, integration, evaluation and testing of software [29][5][22]. To support the model-driven development of a product line it is not sufficient to use a model-driven development tool for class modeling. It is necessary to use an explicit model for each task on each level of abstraction. This leads to the following problems that have to be solved:

- **Model driven development process for product lines:** When developing a model-driven SCM product line, an explicit process description is necessary. But how does a model-driven development process for product lines look like? Can existing product line development processes be used and which steps are necessary to integrate model-driven development?
- **Executable domain modeling language:** Existing product lines are already described by various product

line specific languages on different levels of abstraction. One level is the domain model. Which language can be used to describe the domain model? No existing language is executable in a way, that a graphical high level modeling language is transformed automatically into executable source code. But this property is elementary for model-driven product line development. So, which executable high level graphical modeling language can be adopted as domain modeling language?

- **Selection of a commonality/variability model:** Various models for expressing variabilities and commonalities of products exist already. Which language can be used for a model-driven development process?

- **Mapping from the variability/commonality model onto the executable domain model:** There exist already models for describing single software systems that are executable. But they lack the concepts of variability and commonality. How can these concepts be integrated into existing executable models?

- **Support to design a product line friendly architecture for the domain model**: The architecture of the domain model has to support the configuration of a specific software system out of the domain model. But current languages for architecture specification are only used to describe single applications – not software product lines. Which existing architectural modeling language is adequate for product line architectures? And which architectural concepts (i.e. loose coupling) work especially well for product lines?

- **Support to keep domain, commonality/variability and architectural model consistent:** The model-driven development process will use different models: the commonality/variability model, the architectural model and the domain model. Can existing techniques be used to support consistency between an executable domain model and a commonality/variability model?

3 Related and prior work

In the SCM domain, a few approaches have been developed during the last years to build configurable and adaptable SCM systems. *Whitehead et al.* [31], [32], [33] propose a solution to create SCM systems based on containment modeling. The version models used for the Bamboo system are defined in an extended ER data model (Containment Modeling Framework). An SCM system is generated from these models. However, the model only defines the static structure, but not the behavior of the resulting system (the behavior has to be programmed in Java). Furthermore, no feature model is provided with Bamboo and also no reuse among the CMF models is supported.

In his PhD thesis *Kovse* [18], [19] investigates a product line approach to the model-driven development of versioning systems. This thesis was carried out in the context of database systems rather than SCM systems. The user of the product line defines a version model as a UML profile; stereotypes and tagged values are used to parameterize the behavior of modeling elements. The behavior is programmed in Java (i.e. no executable domain model).

Van der Hoek and *van der Lingen* describe an approach to a pluggable infrastructure for modular configuration management policy composition [26], [27]. The goal of their work is to provide a system, that allows reuse of CM policies (i.e., the rules by which a user evolves artifacts stored in a CM system). The description of the proposed solution is on a very coarse-grained level and no information is given about the design of the single modules and their dependencies.

All described approaches are model-driven only to a limited extent. Whitehead and Kovse only use models to describe the static structure of the system, whereas no information about models is given in the work of van der Hoek. Also, they do not discuss the relationships between the single components of the SCM system, or how decoupling of the components was achieved.

In the domain of model-driven development the vast majority of the available tools only support code generation from class models, for example EMF [9]. There are also some approaches to support generation of code from behavioral models like Fujaba [34], MOFLON [1] and Tiger [10][11][24]. Fujaba [34] is an object-oriented CASE tool with sophisticated code generation facilities which has been applied in numerous projects for quite a number of years. Through its story diagrams, Fujaba supports programming with graph transformation rules. However, it only supports modeling on the level of class- and story diagrams (story diagrams are similar to interaction overview diagrams in UML 2.0 [20], see figure 2, p.5). It lacks support for modeling in the large [4] (e.g. by providing package diagrams) and it has no support for modeling product lines by specifying feature models. MOFLON is a fork of the Fujaba core which is more compliant to the MOF 2.0 standard. It adds basic support for package diagrams. None of the above tools support variability models or modeling a system on the architectural level.

Feature modeling [17][2] emerged to be the state-of-the art approach in variability modeling. Several approaches have been made to map feature model elements to model elements, for example by *Czarnecki* et al. [23], [2], [8] and *Heidenreich* [15], [14], [16]. However, these approaches

only support a mapping of features to static model elements. No mapping to behavioral models is provided.

4 Research hypothesis

The SCM domain is characterized by a large number of systems with more or less similar features incorporating hard-wired version models which have been implemented with considerable effort. A *modular and model-driven product line for SCM* promises following benefits:

1. *Version models* are defined *explicitly* rather than implicitly in the code. This makes it easier to communicate and reason about version models.
2. Modeling comprises both *structure* and *behavior*. Furthermore, behavioral models are executable.
3. Productivity is improved by replacing programming with the creation of *executable models*.
4. Version models are not created from scratch. Rather, reuse is performed on the modeling level by following a *product line* approach [5].
5. The product line is based on a *model library* which is composed of reusable and loosely coupled modules.

We claim that these benefits will ultimately **increase productivity** when it comes to creating new SCM systems, by utilizing commonalities, reusing components and using executable models. The approach will also **facilitate the development process** by making implicit properties explicit and using expressive graphical modeling languages.

Looking at the problem description it becomes apparent that it is not enough to apply existing techniques from product line or model-driven development. Instead, it is necessary to develop a new approach to model-driven development of product lines with a unique process description, executable models which are able to express commonality and variability, as well as tools supporting this process. This approach has to integrate fundamental concepts and successful methods of both the model-driven development and software product line domain.

5 Proposed solution

A model-driven development process for software product lines implies that there is a model for each task on each level of abstraction. In our proposal three models exist: one specifying the commonalities and variabilities, one defining the system architecture and one modeling the domain. Furthermore, the term "model-driven" implies that there is an editor for each of these models as well as an explicit description how these models are interconnected. To experiment and eventually show the feasibility of our process we use it do develop a **mod**ular, **mod**el-driven **s**oftware **c**onfiguration **m**anagement system called **MOD2-SCM**.

To solve the sketched problems, we propose the following solutions:

- **Model driven development process for product lines:** An existing product line process has to be adapted for model-driven development or even to be created from scratch. We created a first proposal which is depicted in figure 1. The process is based on [29], [5], and [22] and distinguishes between *Domain Engineering* and *Application Engineering*. The first area focuses on software for the domain (i.e. the product line as a whole): First the domain is analyzed to generate a *feature model*. Second an *executable domain model* is developed out of it. Application Engineering concentrates on the development of a single application (i.e. an instance of the product line): First a *configuration* that meets the application's requirements is created out of the *feature model*. Then the *configuration* is used to configure the *executable domain model* to obtain the *configured SCM system*. This process is based on the processes suggested in [29][5][22]. But instead of only partially generating the components, they are fully generated and the variants are selected even before generation (cf. [22]).
- **Executable domain modeling language:** Currently we are using Fujaba to model the components of our SCM systems. This is done by specifying UML class diagrams for the static structure and by using story diagrams to describe the behavior. A **story diagram** is an activity diagram with two kinds of nodes: Statement activities and story patterns. The first consists of a fragment of Java code, allowing for seamless integration of textual and graphical programming. The latter is a communication diagram composed of objects and links. Furthermore, objects may be decorated with method calls. Elements with dashed lines represent optional parts of story patterns. A crossed element means that the story pattern may be applied only when the respective element does not exist. In addition to method calls, a story pattern may describe structural changes: Objects and links to be created or deleted are decorated with the stereotype `<<create>>` (green color) or `<<destroy>>` (red color), respectively. Furthermore, `:=` and `==` denote attribute assignments and equality conditions, respectively.

  Figure 2 shows a story diagram that adds a new version to a single-dimensioned version history (i.e. a kind of linked list). The first activity is a statement activity that retrieves the ID of the previous version

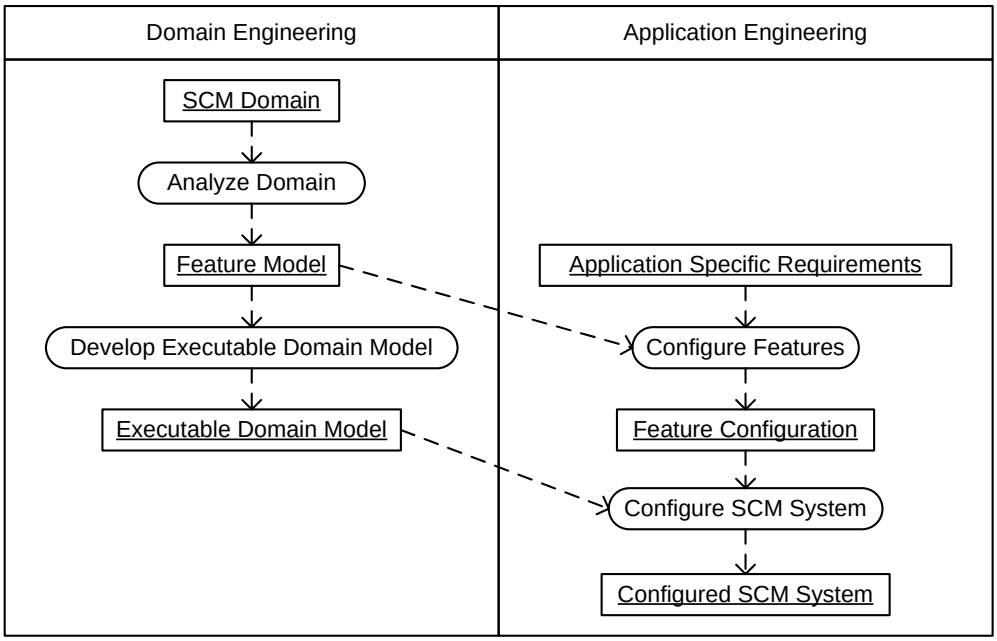

Figure 1. Current MDD product line process.

(`predID`) out of the `context`-Parameter. The second is a story pattern. It creats a `newVersion` and adds it as last element to the `versionSet`. If a `lastVersion` exists already (note this object is optional) whose `versionID` equals the `predID` it becomes the predecessor of the `newVersion`. Ultimately the `content` is added via method call to the `newVersion`. Depending on the outcome of this pattern (successful match or failure) the control flow continues to a stop node that either returns the new `versionID` or `null`.

The Fujaba code generation [13] is used to do the model to code transformation in the *Configure SCM system* step. By extending the **Fujaba model with feature annotations** we introduce the notion of commonality/variability, so the Fujaba models become *executable domain models*. At first we plan to use feature configurations as input for a Fujaba code generation preprocessor to configure the generated code. Later this can be expanded to a complete domain to application model transformation.

- **Selection of a commonality/variability model:** The process starts with a *domain analysis* of the domain the product line has to be created for. The result leads to a *feature model*. We propose **feature models** (see figure 3, p.6), because they are on a high level of abstraction and have turned out as state-of-the art for commonality/variability analysis for product lines.

- **Mapping from the variability/commonality model onto the executable domain model:** By linking the feature annotations from the *Fujaba domain model* to the *feature model* we are able to configure the domain model. Whenever a feature is set in a *feature configuration* the annotated model elements generate code – if the feature is not selected no code is generated. In Fujaba even the behavioral model elements can be annotated so we expect that we are able to do a much more fine-grained product configuration than structure-based mappings.

- **Support to design a product line friendly architecture for the domain model**: We already showed why **UML package diagrams** are important on the architecture level to support modeling in the large [4]. At the moment we use them to describe the system architecture and to analyze the dependencies between the different packages (which is a crucial task to maintain the goal of loosely-coupled components). Currently, we investigate if **UML component diagrams** are a better choice to design a product line friendly architecture.

- **Support to keep domain, commonality/variability and architectural model consistent:** To support a model-driven development process, at least one **validator**-tool, which checks a model against directly related models, is required or at most one **transformer**-tool, which creates a directly related model. Our goal is to create a tool chain to support the model-driven de-

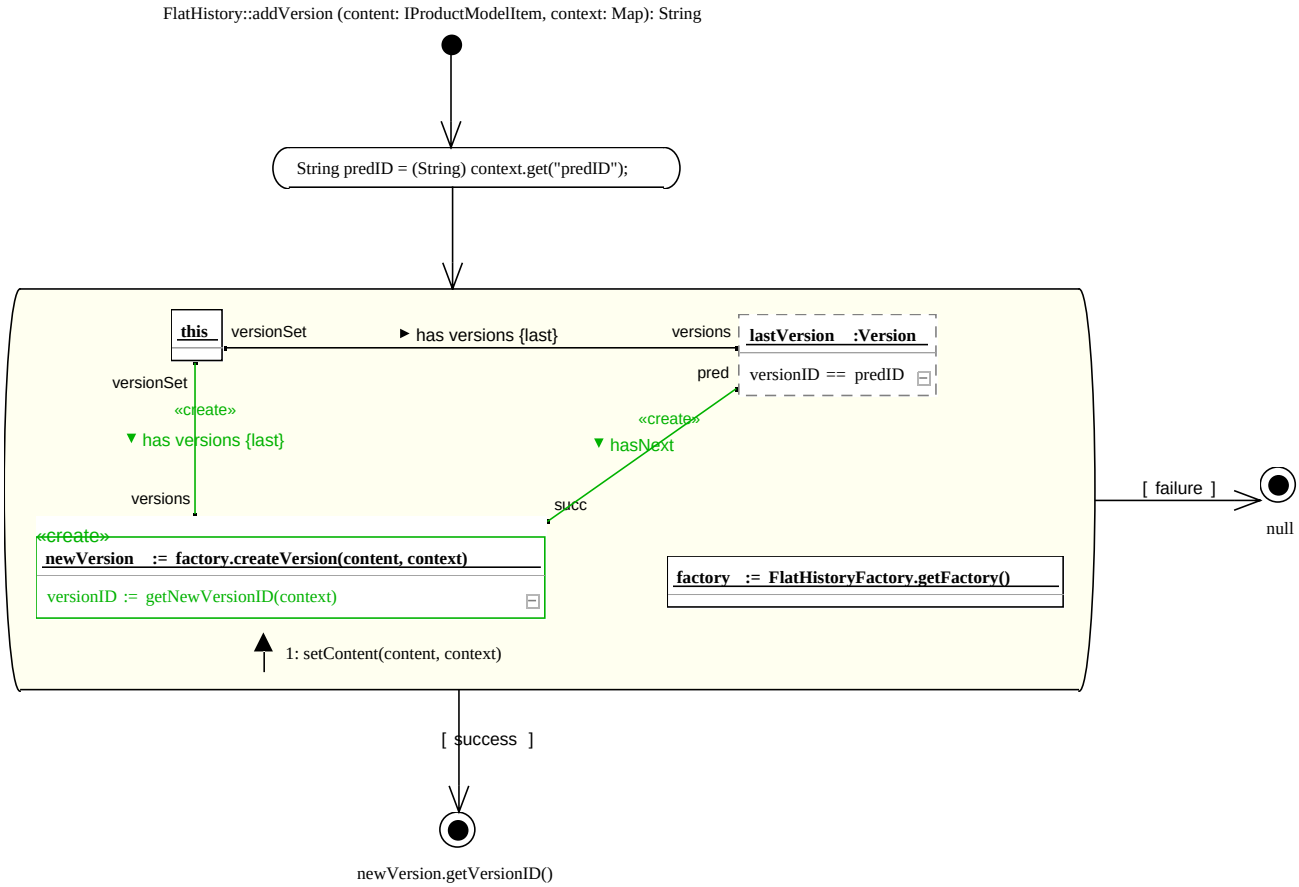

Figure 2. Story diagram for adding a new version to a single-dimensioned history

velopment process. Finally, we will use this tool chain to develop a product line for SCM systems as evaluation of our approach.

For testing applications [5] we intend to use unit tests. A research group of the TU Darmstadt is investigating this task in the model-driven context [21] and we hope to be able to apply the outcome of their research to complete our model-driven process for software product lines.

6 Contributions

The development of our model-driven product line for SCM systems will have several benefits. First of all, we will explore a model-driven development process for developing software product lines. We will use and evaluate different tools that are necessary to carry out the single steps within this process. Furthermore, we can identify possible gaps in the provided tool chain and try to fill them.

As an additional outcome of our work we will get new insights into the interoperability of existing models, for example feature models and UML models on different levels of abstraction. The outcome of our work will also provide informations about the feasibility of the model-driven development approach in the domain of SCM and in general.

By analyzing existing models on different levels of abstraction, we will also get new insights into their expressiveness in terms of describing variability and commonality. Since MOD2-SCM is based on explicit models, we can provide new insights into SCM systems, especially on the coupling and dependencies of the different modules.

We will contribute a modular and configurable system to the SCM domain. Thus, providing a product line for SCM systems will help building new SCM systems with minimal effort.

By using a variability model, the SCM product line can be used to manage all possible configurations of the system. In other words, that would mean configuration management for SCM systems.

7 Current progress

So far, several tasks of the selected product line process are completed or addressed. We finished the tasks *domain understanding* and *requirements engineering* and created the appropriate models to capture our observations.

Also, we already designed a model library for SCM systems which is composed of loosely-coupled components. These components realize the variation points which have been identified in the previous tasks [3], see the feature diagram in figure 3. At the moment the model library supports different kinds of development histories (base version storage without predecessor / successor relationships, one-level history where versions are arranged into a version graph without further grouping and a two-level history, where the graph is composed of branches, each of which consists of a sequence of revisions), different ways to store the contents of versions (baselines, forward deltas, backward deltas and mixed deltas) and two different product models (file system and use-case diagrams). A product model for EMF ecore models is currently under development.

During that part we started to investigate if package diagrams are required on the architectural level in terms of analyzing dependencies between the single components [4]. We already implemented a package diagram editor which supports validation of the class diagram by analyzing the import dependencies on the given package diagram, thus helping the modeler to analyze the coupling between model elements. Besides analyzing and validating the modularity of our system, we also expect that the modeling on the architectural level also supports constructing the final product line, by getting an overview about the dependencies between elements realizing different variation points. The modular architecture of our MOD2-SCM system was built using that package diagram editor.

In future work we will analyze if using component diagrams is a reasonable choice during the process of architecture definition in terms of expressing commonality and variability of a product line. Also, we research the interplay between the different models on each level of abstraction and we will provide a tool chain that supports the model-driven development of a product line for SCM systems.

8 Research methods

During our research we used different methods and formalisms to achieve our goal of creating a model-driven product line for SCM systems. In the beginning, we analyzed the SCM domain using feature modeling [17][2]. The commonalities and variabilities explored during the task domain analysis were modeled using feature diagrams. Modeling on the architectural level was done using UML 2.0 package diagrams [20]. Package diagrams were used to

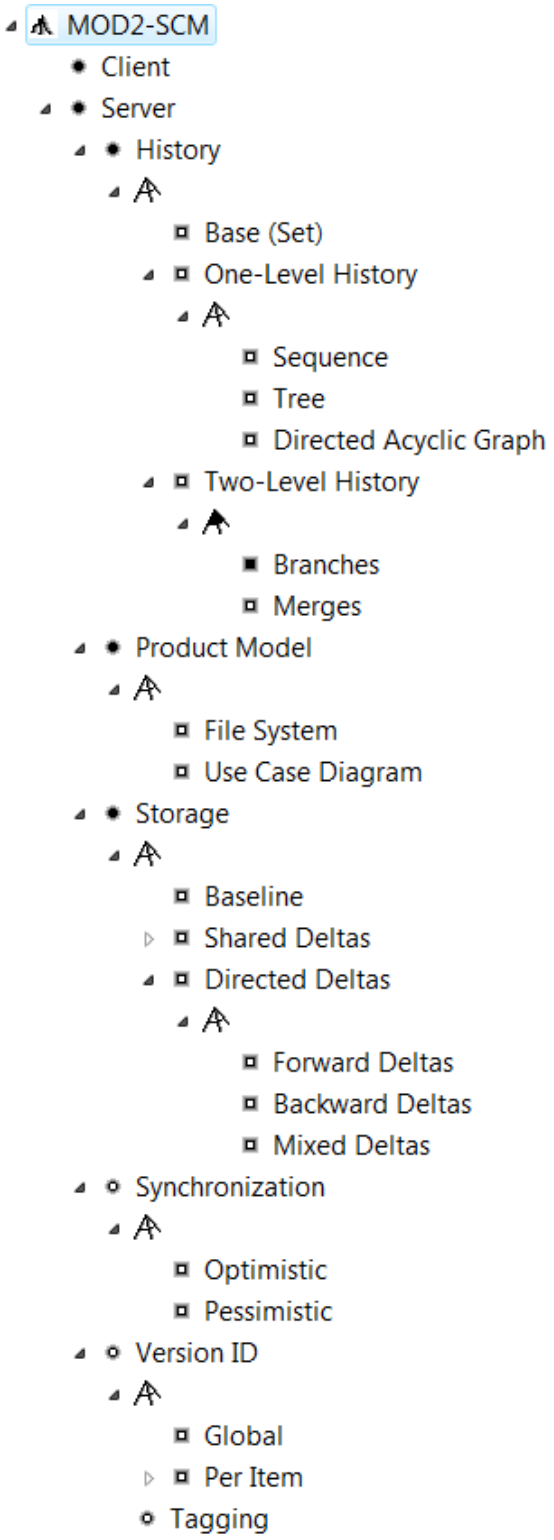

Figure 3. Feature diagram of the variation points that are realized in MOD2-SCM so far.

ensure minimal dependencies between the single packages that are used to realize the variation points of the product line.

To develop the system model we strictly followed the approach of model-driven development. We used the CASE tool Fujaba [34], since it is the only tool which allows to generate executable code from behavioral models. System modeling was done using class diagrams for the static structure and story diagrams for modeling the behavior.

9 Evaluation

To evaluate the feasibility of our approach, we will build a demonstrator, which allows the generation of SCM systems with different properties out of a model library. The development of the demonstrator will be carried out using our model-driven process for product lines.

In a case study, another demonstrator which covers a small part of the SCM system will be build using conventional programming and the conventional product line approach. The required effort will be compared to the effort needed to realize the same system using our model-driven approach.

Last but not least we will evaluate the coupling between the single components of our MOD2-SCM system. This will help us to find out into which extent it is possible to separate the components of SCM systems.

10 Conclusion

In this paper we presented our ideas to create a model-driven process for developing software product lines. So far model-driven development is used to build software systems. We will apply model-driven engineering to the development process of a software product line.

We implement the domain model using Fujaba, which allows the generation of code out of behavioral models. During the development of the domain model, we focused only on a part of the SCM domain (mainly version control) due to the high complexity of this domain. Since the model is the most interesting part for our research, we do not put much effort in creating sophisticated user interfaces. The goal of the development is to build demonstrators, which will show the core functions of e.g. CVS or Subversion.

To express the commonality/variability, we decided to use feature diagrams. Currently we are investigating the mapping of model elements of the commonality/variability model to the executable domain model.

Finally we will investigate, which language can be used to describe a product line architecture for the domain model, and we will focus our research on mechanisms to keep the models on the different levels of abstraction consistent.

References

[1] C. Amelunxen, A. Königs, T. Rötschke, and A. Schürr. Moflon: A standard-compliant metamodeling framework with graph transformations. In A. Rensink and J. Warmer, editors, *Model Driven Architecture - Foundations and Applications: Second European Conference.*, volume LNCS 4066, pages 361–375, Genova, Italy, October 2006 2006. Springer.

[2] M. Antkiewicz and K. Czarnecki. Featureplugin: Feature modeling plug-in for eclipse. In *OOPSLA '04 Eclipse Technology eXchange (ETX) Workshop*, Vancouver, British Columbia, Canada, Oct. 24-28 2004. ACM.

[3] T. Buchmann, A. Dotor, and B. Westfechtel. Model-driven development of software configuration management systems. submitted for publication.

[4] T. Buchmann, A. Dotor, and B. Westfechtel. Experiences with modeling in the large with fujaba. In U. Assmann, J. Johannes, and A. Zndorf, editors, *Proceedings of the 6th International Fujaba Days*. University of Dresden, University of Dresden, 2008.

[5] P. Clements and L. Northrop. *Software product lines: practices and patterns*, volume 0201703327. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[6] B. Collins-Sussman, B. W. Fitzpatrick, and C. M. Pilato. *Version Control with Subversion*. O'Reilly & Associates, Sebastopol, California, 2004.

[7] R. Conradi and B. Westfechtel. Version models for software configuration management. *ACM Computing Surveys*, 30(2):232–282, June 1998.

[8] K. Czarnecki and M. Antkiewicz. Mapping features to models: A template approach based on superimposed variants. In *GPCE 05*, 2005.

[9] Eclipse Foundation. *The Eclipse Modeling Framework (EMF) Overview*, 2009. `http://www.eclipse.org/modeling/emf/` – last visited: 03/02/2009.

[10] K. Ehrig, C. Ermel, S. Hänsgen, and G. Taentzer. Generation of visual editors as eclipse plug-ins. In D. F. Redmiles, T. Ellman, and A. Zisman, editors, *ASE*, pages 134–143. ACM, 2005.

[11] C. Ermel, K. Ehrig, G. Taentzer, and E. Weiss. Object oriented and rule-based design of visual languages using tiger. *Electronic Communications of the EASST*, 1:1–12, 2006.

[12] J. Estublier and R. Casallas. The Adele configuration manager. In W. F. Tichy, editor, *Configuration Management*, volume 2 of *Trends in Software*, pages 99–134. John Wiley & Sons, New York, 1994.

[13] L. Geiger, C. Schneider, and C. Reckord. Template- and modelbased code generation for mda-tools. In H. Giese and A. Zndorf, editors, *Proceedings of the 3rd International Fujaba Days*. Universitt Paderborn, 2005.

[14] F. Heidenreich, I. Şavga, and C. Wende. On controlled visualisations in software product line engineering. In *Proceedings of the 2nd International Workshop on Visualisation in Software Product Line Engineering (ViSPLE 2008), collocated with the 12th International Software Product Line Conference (SPLC 2008)*, Sept. 2008. To appear.

[15] F. Heidenreich, J. Kopcsek, and C. Wende. Featuremapper: Mapping features to models. In *Companion Proceedings of the 30th International Conference on Software Engineering (ICSE'08)*, pages 943–944. ACM, May 2008.

[16] F. Heidenreich and C. Wende. Bridging the gap between features and models. In *2nd Workshop on Aspect-Oriented Product Line Engineering (AOPLE'07) co-located with the International Conference on Generative Programming and Component Engineering (GPCE'07)*, 2007. URL http://www.softeng.ox.ac.uk/aople/.

[17] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.

[18] J. Kovse. *Model-driven development of versioning systems*. PhD thesis, Technische Universitt Kaiserslautern, Kaiserslautern, 2005.

[19] J. Kovse and T. Härder. Model-driven development of versioning systems: An evaluation of different approaches. Technical report, University of Kaiserslautern, Kaiserslautern, 2005.

[20] OMG. *OMG Unified Modeling Language (OMG UML), Superstructure*. OMG, November 2007. Version 2.1.2.

[21] S. Oster, A. Schürr, and I. Weisemöller. Towards software product line testing using story driven modeling. In U. Assmann, J. Johannes, and A. Zndorf, editors, *Proceedings of the 6th international Fujaba Days*. University of Dresden, University of Dresden, 2008.

[22] K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, Berlin, Germany, 2005.

[23] M. Stephan and M. Antkiewicz. Ecore.fmp a tool for editing an instantiating class models as feature models. Technical report, University of Waterloo, 2008.

[24] Technische Universität Berlin. *TIGER Project*, 2005. `http://tfs.cs.tu-berlin.de/tigerprj/`.

[25] W. F. Tichy. RCS – A system for version control. *Software: Practice and Experience*, 15(7):637–654, July 1985.

[26] R. van der Lingen and A. van der Hoek. Dissecting configuration management policies. In *Software Configuration Management*, pages 177–190, 2003.

[27] R. van der Lingen and A. van der Hoek. An experimental, pluggable infrastructure for modular configuration management policy composition. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 573–582, Washington, DC, USA, 2004. IEEE Computer Society.

[28] J. Vesperman. *Essential CVS*. O'Reilly & Associates, Sebastopol, California, 2006.

[29] D. M. Weiss and C. T. R. Lai. *Software product-line engineering: a family-based software development process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[30] B. A. White. *Software Configuration Management Strategies and Rational ClearCase*. Object Technology Series. Addison-Wesley, Reading, Massachusetts, 2003.

[31] E. J. Whitehead and D. Gordon. Uniform comparison of configuration management data models. In B. Westfechtel and A. van der Hoek, editors, *Software Configuration Management: ICSE Workshops SCM 2001 and SCM 2003*, LNCS 2649, pages 70–85, Portland, Oregon, 2003. Springer-Verlag.

[32] J. E. Whitehead. Uniform comparison of data models using containment modeling. In *HYPERTEXT '02: Proceedings of the thirteenth ACM conference on Hypertext and hypermedia*, pages 182–191, New York, NY, USA, 2002. ACM Press.

[33] J. E. Whitehead, G. Ge, and K. Pan. Automatic generation of version control systems. Technical report, University of California, 2004.

[34] A. Zündorf. Rigorous object oriented software development. Technical report, University of Paderborn, Germany, 2001.