Model-Driven Product Consolidation into Software Product Lines

Benjamin Klatt, Klaus Krogmann
FZI Forschungszentrum Informatik
Haid-und-Neu-Str. 10-14, 76131 Karlsruhe, Germany
{klatt,krogmann}@fzi.de

1 Introduction

Software Product Lines (SPL) are an established concept for software vendors to reach a high level of reuse and customisation at the same time. Due to cost and time constraints, especially in evolving domains, reactive approaches (extending existing SPL) and extractive approaches (new SPL from multiple products) dominate proactive approaches (SPL first) in practise [1]. However, as easy it is to copy and customise products, as challenging it is to consolidate them into a common product line again. Major challenges are the necessary change comprehension, feature identification, and refactoring decisions.

To cope with the challenges, we propose to fill the gap between reverse-engineered low-level software models (e.g., Abstract Syntax Trees (AST)) and capability-level feature models (FM) for SPL engineering. We introduce an intermediary Variation Point Model (VPM) that links those two ends, and captures design decisions in terms of cardinality between features and variation points as well as choosen variability realisation techniques. We further propose an incremental, semi-automatic creation, refinement, and refactoring of the VPM and FM supported with model-based analysis for difference-comprehension of the product copies under study. Although researchers, such as Alves et al. [1], have identified that variability-aware SPL refactoring techniques are required, a reasonable support to combine the two worlds of implementation and feature models is still missing.

Our work focuses on the creation of SPLs from customised product copies with a common initial code-base, comparable to the appraoch by Koschke et al. [8]. However, our approach does not require a pre-existing module-architecture as the approach by Kosche et al. to use their extended reflexion method. Furthermore, we aim to take custom SPL requirements ('SPL Profile') into account, such as the intended maturity level and the preferred variation point design and realisation techniques.

As part of our overall process [7], we propose a model-driven approach with i) established models on both ends, ii) winning feature-to-code-traces from integrated models and process, and ii) taking individual SPL requirements into account ('SPL Profile').

2 Model-Driven Process

Whether an existing SPL should be extended with an ad-hoc customised product (reactive approach) or a new SPL should be created by consolidating multiple products (extractive approach) – implementation-level models of the existing products should be extracted in both cases. Such models contain the structural elements of the software, referable by other models and linking the original sources. As shown in Figure 1, the goal is to gain a SPL feature model linked with the software entity model to trace the related code contributions and vice versa.

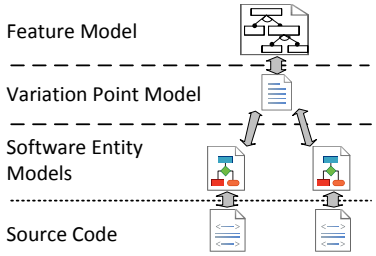

Figure 1: Code to Feature Models

In addition to general trace approaches (e.g., Riebisch et al. [11]), more specific support to build models and links is provided by diff'ing- and software-analysis. Their results are represented as models, integrated with the reverse-engineered software entity models and support the change comprehension of different product variants. These trace links can also be used to provide feedback when designing features of an SPL. For example, if an SPL engineer decides to merge two features, a warning could be provided if this would affect two unrelated code areas. The following sections present the process steps for the creation of the proposed linked models. The already mentioned SPL Profile influences all of these steps, e.g., if one prefers to realise variation points with exchangeable components, he might be interested in a more coarse-grain clustering as if he would prefer to use a preprocessing on a statement level.

1. Extract Implementation Level Models: Due to the overall goal of supporting a software refactoring (i.e. introduce an SPL), we need detailed models of the software products with mappings to the analysed code. Abstract Syntax Trees (AST) provide directed graph representations of the softare entities, such as the OMG standard [10] implemented for the Eclipse Modeling Framework (EMF) including tools support by the MoDisco project [2].

2. Difference Analysis: The differences between the implementations of two software products can be

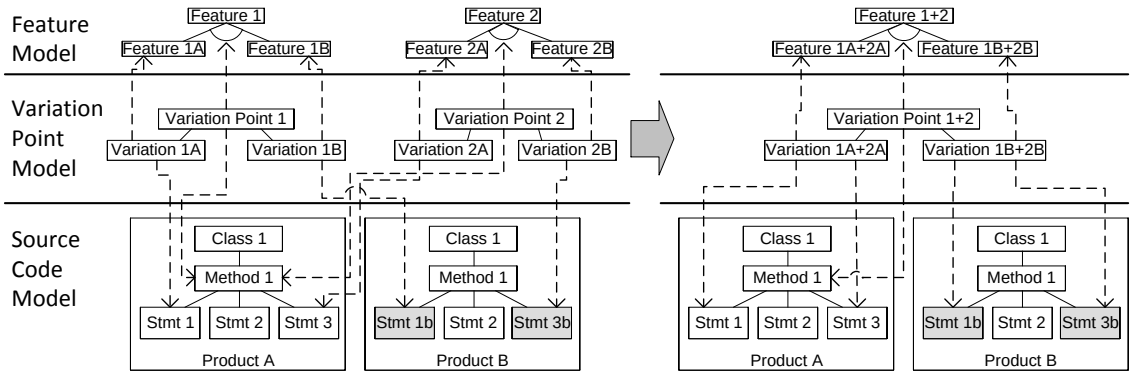

Figure 2: Variation Point Merge Example

identified by either comparing the current state of the software artifacts or by analysing their incremental change history (e.g., from version control systems or by tracking edit operations). We prefer the former approach to not exclude scenarios without access to such a history. However, even the latter approach is valid for some scenarios. With both approaches, the goal is to derive a model of the differences in the ASTs with references to the differing elements in the software models and their type of change. Existing research, such as the one by Fluri et al. [6], has shown that software difference analysis taking software structures (e.g., method and class declarations) into account provides advantages over simple text-based differences. Fluri et al. have also published a catalog of change types which is a good starting point for such a difference model.

With regard to the EMF-based implementation of the standardised OMG AST model, such a difference analysis can be built upon the EMF compare project, which provides an extensible infrastructure to create such a model and the according analysis.

3. Analysis for Change Comprehension: Existing analysis approaches for change comprehension have two goals: Reduce the amount of information to be reviewed and find relationships not explicitly represented or obvious in the software. We propose to use a combination of change analysis (e.g., Fluri et al. [6]), software architecture reverse engineering (e.g., Chouambe et al. [4]), and specific approaches such as feature location (e.g., static or dynamic program graph analysis as done by Rajlich et al. [3], Wilde et al. [12], or Graaf et al. [5]) to identify, for example, related code changes which might later serve as a variable product feature.

4. Feature Mapping: The target feature model describes the possible design space for the products that can be realised with an SPL. A software engineer responsible for the SPL has to decide which features should be realised in the SPL and how they are mapped to the original code. This decision should be based on the initial software entity models, the difference model and the results of the change-comprehension analysis. This mapping is specified within a Variation Point Model (VPM) describing the variation points in the software implementation as well as the available alternatives and their representation in the code.

We propose an incremental creation and enhancement of these mappings. An initial mapping can be created automatically: differing code elements are linked by alternative variations and their parent element is linked by a variation point. The VPM can be semi-automatically refined in the downstream process. For example, as presented in Figure 2, if multiple statements in a class's method differ, the individual variation points of these modifications might be merged to a single variation point and the according variations as well as the referenced feature and sub-features might be merged as well.

Recommendations to support the decision process can be based on i) patterns and relationships detected in the code (analysis models), ii) learning from architects' previous decisions (which variation points have been merged before), and iii) target SPL requirements (e.g., the preferred implementation mechanism, such as dependency injection, configurations, or code generation).

Alves et al. [1] formalised valid, feature-aware refactorings for SPLs. Their approach is complementary to ours and can be considered as a refinement step to refactor initially varying products to a homogenous product line. They do not provide support for identifying code differences contributing to a feature. Instead, they assume feature models to be derived from documentation or manual code examination. However, they state that model-driven support is desirable.

3 Assumptions, Limitations, Outlook

Our approach assumes existing software products derived from a common intial code base. Due to this, we do not support consolidating independently developed products, such as studies by Koziolek et al. [9]. We assume the degree of changes in the product copies will have an influence on the results of our approach, which needs to be evaluated in future case studies.

A prototype of the proposed approach is currently

under development. We further work on identifying approaches for the change comprehension, assistance for variation point merging and model transformations for the downstream SPL refactoring process.

References

[1] V. Alves, R. Gheyi, T. Massoni, U. Kulesza, P. Borba, and C. Lucena. Refactoring Product Lines. In *GPCE'06*. ACM.

[2] H. Bruneliere, J. Cabot, F. Jouault, and F. Madiot. MoDisco: A generic and extensible framework for model driven reverse engineering. In *ASE'10*.

[3] K. Chen and V. Rajlich. Case study of feature location using dependence graph. In *IWPC'2000*. IEEE.

[4] L. Chouambe, B. Klatt, and K. Krogmann. Reverse Engineering Software-Models of Component-Based Systems. In *CSMR'08*. IEEE.

[5] B. Cornelissen, B. Graaf, and L. Moonen. Identification of variation points using dynamic analysis. In *R2PL'2005*.

[6] B. Fluri, M. Wuersch, M. Pinzger, and H. Gall. Change Distilling: Tree Differencing for Fine-Grained Source Code Change Extraction. *IEEE TSE*.

[7] B. Klatt and K. Krogmann. Towards Tool-Support for Evolutionary Software Product Line Development. In *WSR'11*.

[8] R. Koschke, P. Frenzel, A. P. J. Breu, and K. Angstmann. Extending the reflexion method for consolidating software variants into product lines. *Software Quality Journal*, 2009.

[9] H. Koziolek, R. Weiss, and J. Doppelhamer. Evolving Industrial Software Architectures into a Software Product Line: A Case Study. In *Architectures for Adaptive Software Systems (LNCS)*. Springer, 2009.

[10] OMG. Abstract Syntax Tree Metamodel (ASTM). Technical report, 2011.

[11] M. Riebisch, S. Bode, Q.-U.-A. Farooq, and S. Lehnert. Towards Comprehensive Modelling by Inter-model Links Using an Integrating Repository. *ECBS'11*.

[12] N. Wilde and M. C. Scully. Software reconnaissance: Mapping program features to code. *Journal Of Software Maintenance Research And Practice*, 1995.