Acta Informatica Pragensia, 2016, 5(2): 138–143
DOI: 10.18267/j.aip.90
Peer-reviewed paper

A DSL-based Approach to Product Derivation for Software Product Line

Nesrine Lahiani[*], Djamal Bennouar[†]

Abstract

Product derivation is an important part of the Software Product Line (SPL) development process. The quality of a product derivation process has a direct impact in decreasing software product costs and time-to-market. In this paper, we present an approach that represents the SPL with a set of integrated models and automatically derives executable products with model transformations. We combine SPL and Model-Driven Engineering (MDE) into a comprehensive and extremely effective framework in order to get advantages of both techniques. In order to evaluate the feasibility of our approach, we have designed and implemented it using existing and available technologies.

Keywords: Product derivation, Software product line, Domain specific language, Model-driven engineering.

1 Introduction

A software product line (SPL) is as a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way (Clements & Northrop, 2002). The SPL engineering process (see Figure 1) consists of two major phases: a) domain engineering for analysing the commonality and variability between members of the product line and establishing reusable SPL models, and b) application engineering for deriving manually or automatically an individual product from these reusable models instead of starting from scratch.

We focus in this paper at application engineering known also as Product Derivation (PD). PD has been defined in many different ways, McGregor (2009) defines it by "Product derivation is the focus of a software product line organization and its exact form contributes heavily to the achievement of targeted goals". Deelstra, Sinnema & Bosch (2005) define product derivation by, "A product is said to be derived from a product family if it is developed using shared product family artifacts. The term product derivation therefore refers to the complete process of constructing a product from product family software assets".

[*] Department of Informatics, Université Saâd Dahlab de Blida, B. P. 270, Route de Soumâa, 09000 Blida, Algeria
✉ lahiani.nesrine@gmail.com

[†] Department of Informatics, Université de Bouira, Rue Drissi Yahia, Bouira 10000, Algeria
✉ djamal.bennouar@univ-bouira.dz

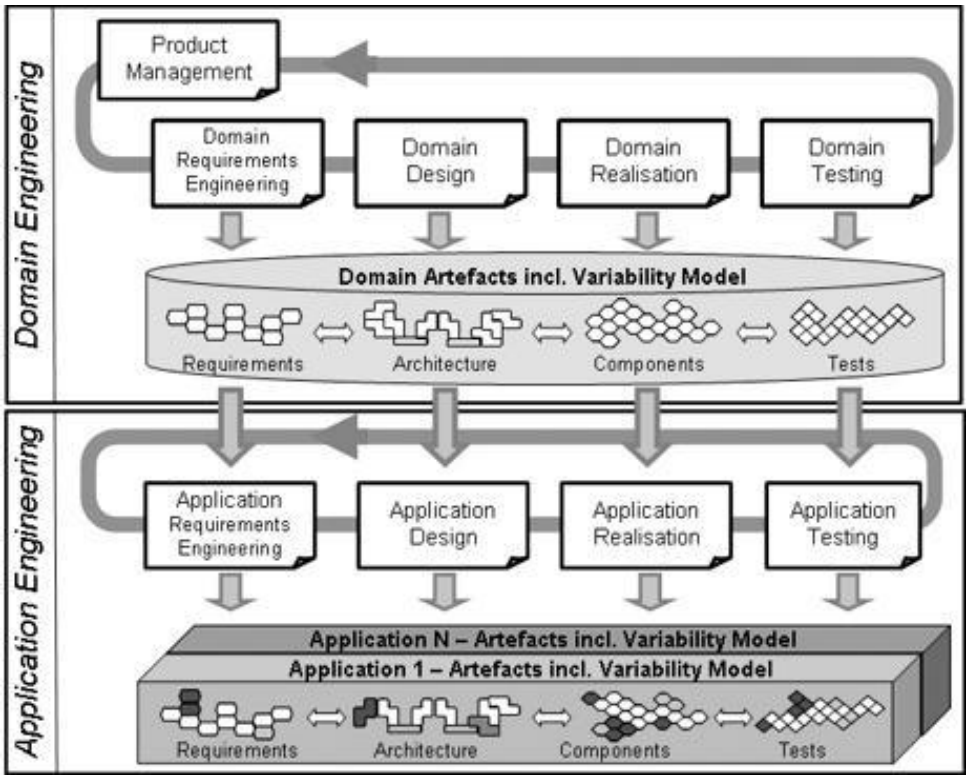

Figure 1. *The software product line engineering framework. Source: (Pohl, Böckle, & van Der Linden 2005).*

In this paper, we propose an approach that supports: (i) modelling the variability; and (ii) deriving product by using existing technologies. We believe that model driven engineering has a prominent role to play in product-line engineering to define their core assets and support product derivation.

The remainder of this paper is organized as follows. Section 2 presents existing research works of product derivation and feature mapping techniques. Section 3 gives an overview of the main elements and functionalities of our approach. Section 4 describes the approach implementation using existing model-driven technologies. Finally, Section 5 presents the conclusions

2 Related works

This section briefly presents related works on product derivation approaches and different feature mapping techniques. *FArM* (Feature-Architecture Mapping) method proposed by Sochos et al. (2006). FArM provides a stronger mapping between features and the architecture. It is based on a series of transformations on the initial PL feature model. During these transformations architectural components are derived, encapsulating the business logic of each transformed feature and having interfaces reflecting the feature interactions.

Tawhid et al. (2011) proposed to derive an UML model of a specific product from the UML model of a product line based on a given feature configuration is enabled through the mapping between features from the feature model and their realizations in the design model. The mapping technique proposed aims to minimize the amount of explicit feature annotations in the UML design model of SPL. Implicit feature mapping is inferred during product derivation from the relationships between annotated and non-annotated model elements as

defined in the UML metamodel and well-formed rules. The transformation is realized in the Atlas Transformation Language (ATL).

ArchFeature a recent work proposed by Gharibi et al. (2016) which is a PLA modelling approach equipped with a graphical environment. ArchFeature integrates feature specification, PLA, and their relationships in a single monolithic architecture model. This is enabled by extending an existing XML-based architecture description language (ADL), xADL that is mostly used for modeling a single system's architecture consisting of components and connections. It includes a graphical modelling environment that can (1) automatically capture, maintain, and visualize the feature-PLA relationship, (2) encapsulate variability modelling from the user, and (3) support automatic derivation of architecture instances from the PLA. ArchFeature is integrated in ArchStudio, an Eclipse-based architecture development platform.

FeatureMapper an Eclipse-based tool proposed by Heidenreich et al. (2008) supports mapping features from a feature model to solution artifacts expressed in EMF/Ecore based languages (e.g. UML2). A feature configuration containing a set of selected features is combined with the mapping model and interpreted by the FeatureMapper transformation component to derive a product model. The negative variability technique is applied for product derivation, and the mapping is done through a separate model.

3 Concepts and motivation

Model-Driven Engineering (MDE) aims at reducing the accidental complexity associated with developing complex software-intensive systems (Schmidt, 2006). A primary source of accidental complexity is the large gap between the high-level concepts used by domain experts to express their problem statements and the low-level abstractions provided by general-purpose programming languages (France & Rumpe, 2006). Manually bridging this gap, particularly in the presence of changing requirements, is costly in terms of both time and effort. MDE approaches address this problem through the use of modelling techniques that support separation of concerns and automated generation of major system artifacts (e.g., test cases, implementations) from models. In MDE, a model describes an aspect of a system and is typically created for specific development purposes. Separation of concerns is supported through the use of different modelling languages, each providing constructs based on abstractions that are specific to an aspect of a system (Cheng et al., 2015).

A Domain Specific Language (Stahl et al., 2006) is formalism for building models: It encompasses a metamodel as well as a definition of a concrete syntax that is used to represent the models. The concrete syntax can be textual, graphical or using other means, such as tables, trees or dialogs. Different DSLs can use the same metamodel while varying in their concrete syntax. The models built with these DSLs will look different, but will all have the same meaning. The meta-model is what the tools care about; whereas the concrete syntax is what the DSL users care about. It is essential, that the concrete syntax can sensibly represent the concepts the DSL is intended to describe.

Incorporating domain-specific concepts and best practices development experience into MDE technologies can significantly improve developer productivity and system quality. A DSL provides a bridge between the (problem) space in which domain experts work and the implementation (programming) space (Cheng et al., 2015).

4 Product derivation process

In this section, we present an overview of our approach for product derivation. It is founded on the principles and techniques of software product lines and model driven engineering. Figure 2 illustrates the main elements of our approach and their respective relationships.

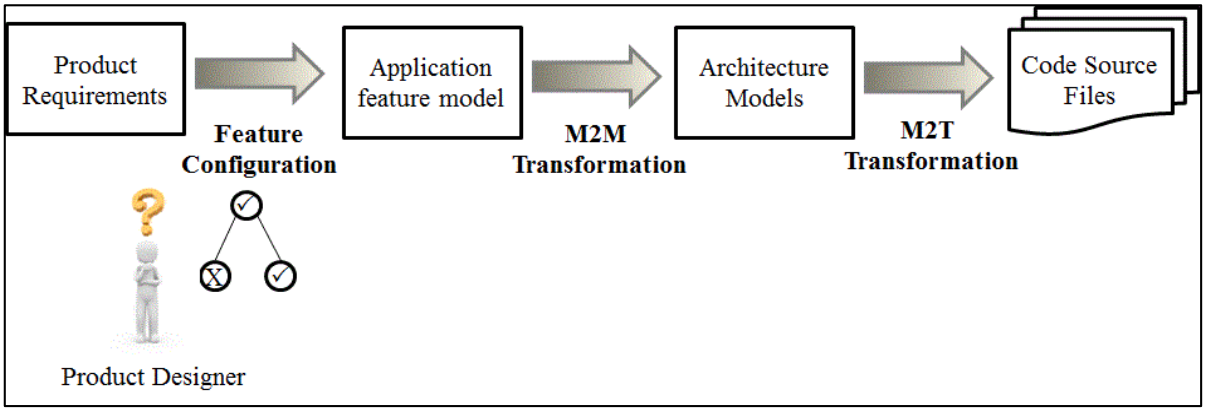

Figure 2. *Overview of proposed approach. Source: Authors.*

Before starting the fourth principal activities, two important inputs are required

- Customer's requirements: to describe what customer wants, its requirements documented in natural language (textual requirements) or by conceptual models (model-based requirements)
- Feature Model: Our feature model is based on (Czarnecki et al., 2005) which focus on identifying external visible characteristics of products in terms of commonality and variability, rather than describing all details of products such as other modelling techniques. Features can be common, optional, or alternative.

The proposed derivation approach uses the mapping of features to architecture model (Lahiani & Bennouar, 2015). Next we briefly explain the activities of the proposed approach. The first activity of our approach is the feature configuration. A feature configuration is a legal combination of features that specifies a particular product. Step1 uses feature models as input to select the feature relevant for customer's requirements to build the product and identify the specific-assets of the product. Once the selection is checked and validated by the product designer the output at this stage is a specialized version of feature model (application feature model). After that, application feature model is considered an input parameter and then is processed by a model-to-model (M2M) transformation written in ATL (Atlas Transformation Language) that creates an Architecture Model which composed of a set of rules and helpers. The rules define the mapping between the source and target model. The helpers are methods that can be called from different points in the ATL transformation. This model describes all components that have to be included to implement this particular Application Feature Model. The model is then processed by a model-to-text (M2T) transformation which generates an equivalent textual configuration implemented using Acceleo language to promote the generation of Java.

5 Conclusion

Derivation of a product from an SPL seems to be an easy step since it's relied on reuse. Actually the product derivation represents one of the main challenges that SPL faces due to time-consuming. In this paper, we intend to reduce the development time of a product by automating the derivation by generating some java code using Acceleo in conjunction with ATL.

The proposed transformation uses Feature-architecture mapping technique by instantiating the initial feature model, an instance of feature model is constructed according to customer's requirements. Then, separate features into two kinds: common and variable. The main idea is to create for each feature a component or a set of components combined in a specific way. Linking these created components together based on the relationships among features in the feature model is the last step of our process.

References

Cheng, B. H., Combemale, B., France, R. B., Jézéquel, J. M., & Rumpe, B. (2015). On the Globalization of Domain-Specific Languages. In *Globalizing Domain-Specific Languages* (pp. 1-6). New York: Springer. doi: 10.1007/978-3-319-26172-0_1

Clements, P., & Northrop, L. (2002). *Software Product Lines: Practices and Patterns.* Boston: Addison-Wesley.

Czarnecki, K., Helsen, S., & Eisenecker, U. (2004). Staged Configuration Using Feature Models. In *Software Product Lines, Proceedings of the Third International Conference SPLC 2004* (pp. 266-283). Berlin: Springer. doi: 10.1007/978-3-540-28630-1_17

Dashofy, E. M., Hoek, A. V. D., & Taylor, R. N. (2005). A comprehensive approach for the development of modular software architecture description languages. *ACM Transactions on Software Engineering and Methodology*, 14(2), 199-245. doi: 10.1145/1061254.1061258

Deelstra, S., Sinnema, M., & Bosch, J. (2005). Product derivation in software product families: a case study. *Journal of Systems and Software*, 74(2), 173-194. doi: 10.1016/j.jss.2003.11.012

France, R., & Rumpe, B. (2007). Model-driven development of complex software: a research roadmap. In *Proceedings of the Future of Software Engineering Symposium* (pp. 37-54). New York: IEEE. doi: 10.1109/FOSE.2007.14

Gharibi, G., & Zheng, Y. (2016). ArchFeature: Integrating features into product line architecture. In *Proceedings of the 31st Annual ACM Symposium on Applied Com*puting (pp. 1302-1308). New York: ACM. doi: 10.1145/2851613.2851764

Heidenreich, F., Kopcsek, J., & Wende, C. (2008). FeatureMapper: Mapping features to models. In *Companion of the 30th international conference on Software engineering* (pp. 943-944). New York: ACM. doi: 10.1145/1370175.1370199

Lahiani, N., & Bennouar, D. (2015). A Software Product Line Derivation Process Based on Mapping Features to Architecture. In *Proceedings of the International Conference on Advanced Communication Systems and Signal Processing*. Tlemcen: University of Abou Bekr Belkaid Tlemcen.

McGregor, J. (2009). Goal-driven Product Derivation. *Journal of Object Technology*, 8(5), 7-19. doi: 10.5381/jot.2009.8.5.c1

Pohl, K., Böckle, G., & van Der Linden, F. J. (2005). *Software product line engineering: foundations, principles and techniques*. Berlin: Springer. doi: 10.1007/3-540-28901-1

Schmidt, D.C. (2006). Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2), 25-31. doi: 10.1109/MC.2006.58

Sochos, P., Riebisch, M., & Philippow, I. (2006). The Feature-Architecture Mapping (FArM) Method for Feature-Oriented Development of Software Product Lines. In *Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems* (pp. 308-318). New York: IEEE. doi: 10.1109/ECBS.2006.69

Stahl, T., Voelter, M., & Czarnecki, K. (2006). *Model-driven software development: technology, engineering, management*. Hoboken, NJ: John Wiley & Sons.

Tawhid, R., & Petriu, D. C. (2011). Product model derivation by model transformation in software product lines. In *Proceedings of the 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops* (pp. 72-79). New York: IEEE. doi: 10.1109/ISORCW.2011.18