The Journal of Systems and Software 83 (2010) 1108–1122

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Structuring the modeling space and supporting evolution in software product line engineering

Deepak Dhungana [a,*], Paul Grünbacher [b,c], Rick Rabiser [b], Thomas Neumayer [b]

[a] Lero – The Irish Software Engineering Research Centre, University of Limerick, Limerick, Ireland
[b] Christian Doppler Laboratory for Automated Software Engineering, Johannes Kepler University, Linz, Austria
[c] Institute for Systems Engineering and Automation, Johannes Kepler University, Linz, Austria

ARTICLE INFO

Article history:
Received 8 February 2009
Received in revised form 3 February 2010
Accepted 11 February 2010
Available online 24 February 2010

Keywords:
Product line engineering
Model evolution
Variability modeling

ABSTRACT

The scale and complexity of product lines means that it is practically infeasible to develop a single model of the entire system, regardless of the languages or notations used. The dynamic nature of real-world systems means that product line models need to evolve continuously to meet new customer requirements and to reflect changes of product line artifacts. To address these challenges, product line engineers need to apply different strategies for structuring the modeling space to ease the creation and maintenance of models. This paper presents an approach that aims at reducing the maintenance effort by organizing product lines as a set of interrelated model fragments defining the variability of particular parts of the system. We provide support to semi-automatically merge fragments into complete product line models. We also provide support to automatically detect inconsistencies between product line artifacts and the models representing these artifacts after changes. Furthermore, our approach supports the co-evolution of models and their respective meta-models. We discuss strategies for structuring the modeling space and show the usefulness of our approach using real-world examples from our ongoing industry collaboration.

1. Introduction and motivation

Many software product lines today are developed and maintained using model-based approaches, e.g., feature-oriented modeling (Kang et al., 1990; Czarnecki and Eisenecker, 1999; Asikainen et al., 2006), decision-based approaches (Dhungana et al., 2007a; Schmid and John, 2004), orthogonal approaches (Bachmann et al., 2003), architecture modeling languages (Matinlassi, 2004; Dashofy et al., 2001), or UML-based techniques (Atkinson et al., 2002; Gomaa, 2005). Tools have been developed to automate domain and application engineering based on models defining core assets and their commonalities and variability (Dhungana et al., 2007c). General purpose variability modeling approaches (Gomaa and Shin, 2002; Sinnema et al., 2004; Dhungana et al., 2007a) allow defining the variability of arbitrary domain-specific assets via meta-models that specify the possible elements of variability models (e.g., types of reusable assets, their attributes, types of dependencies).

No matter which modeling approach is followed, product line engineering (PLE) faces two challenges: (i) developing a single product line model is practically infeasible due to the scale and complexity of today's systems: the high number of features and components in real-world systems means that modelers need strategies and mechanisms to organize the modeling space. (ii) new customer requirements, technology changes, and internal enhancements lead to the continuous evolution of a product line's reusable assets: While product line engineers try to understand and capture the variability of a complex existing system, the reusable assets are frequently changed to meet evolving business needs. Evolution support becomes particularly important in a model-based approach to ensure consistency after changes to meta-models, models, and actual development artifacts. Product line approaches need to treat maintenance and evolution as critical due to the longevity of many systems. Many existing approaches are instead based on the assumption that a product line is fairly stable. However, such stability cannot be taken for granted. PLE should thus treat evolution as the normal case and not as the exception (Dhungana et al., 2008b).

An analysis of development practices of our industry partner Siemens VAI[1], the world's leading steel plant building company, revealed three important issues related to evolution in model-based PLE:

* Corresponding author.
E-mail addresses: deepak.dhungana@gmail.com, deepak.dhungana@lero.ie (D. Dhungana), paul.gruenbacher@jku.at (P. Grünbacher), rabiser@ase.jku.at (R. Rabiser), neumayer@ase.jku.at (T. Neumayer).

[1] http://www.industry.siemens.com/metals/en/.

0164-1212/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2010.02.018

(i) *Modeling space structuring*. The size of many software systems is far beyond the ability of any individual or small group to understand them in detail. This prevents effective coordination because a single individual or small group cannot direct its work and keep all the implementation details in focus (Kraut and Streeter, 1995). To address this challenge the *modeling space has to be structured*, so that large product lines can be managed more easily. This challenge is related to Conway's law (Conway, 1968; Herbsleb and Grinter, 1999) describing dependencies between the communication structure of a development team and the technical structure of a system.

(ii) *Model consistency*. Different parts of the system evolve at different speeds and have to be *kept consistent* with the models describing these parts. Product line assets evolve continuously to address changes such as new customer requirements, technology changes, or refactoring. For example, a large component may be divided into several parts, a component may be moved from one subsystem to another, or new relationships between components may be established. It is therefore essential to understand, model, and maintain the links between the product line's variability models and its asset base. Engineers should be supported in detecting and keeping track of inconsistencies during modeling (Vierhauser et al., 2010).

(iii) *Meta-model evolution*. As pointed out domain meta-models are also subject to evolution. In an effective model-driven development cycle modeling tools and techniques must be adaptable to changing requirements in the problem domain. For instance, the introduction of new asset types or the modification of existing asset types require updating existing models. The domain *meta-models need to co-evolve* with the variability models.

Based on our analysis of current practices and needs of our industry partner, we have developed a model-based approach for defining, managing, and utilizing product lines (Dhungana et al., 2007b; Rabiser et al., 2007). Supporting evolution has been a critical success factor during development. Our approach is based on a simple assumption: A small model is easier to maintain than a large one. Instead of creating a single large product line variability model we use *model fragments* to describe the variability of selected parts of the system. These model fragments also represent the units of evolution in our approach (Dhungana et al., 2008a). The approach meets the demands of real-world development processes as different teams can work on variability model fragments describing the parts of the system they know best.

We have presented our approach to deal with evolution in previous publications (Dhungana et al., 2008b,a; Grünbacher et al., 2009). Here, we further elaborate the underlying issues and present our experiences of applying the tool-supported approach for a real-world product line of Siemens VAI. The company is maintaining a software product line for the automation of continuous casting in steel plants.

In particular we claim three contributions: (i) An approach based on *model fragments* for the decentralized creation and maintenance of product line variability models. (ii) Tool support for the automated detection of changes to *keep models and architecture consistent*. (iii) Tools and techniques facilitating *meta-model evolution* for propagating changes in the domain to already existing variability models.

The paper is organized as follows: Section 2 elaborates on the needs for structuring the modeling space. In Section 3 we describe our model fragment-based approach. Section 4 presents tool support for creating and managing the model fragments, consistency checking, and meta-model evolution. In Section 5 we discuss our experiences of applying the approach at Siemens VAI and discuss strengths and weaknesses of our approach. Section 6 presents related work. Finally, we present conclusions and an outlook on future work in Section 7.

2. Structuring the modeling space

Software evolution is challenged by the fact that development teams require a mix of skills. In many software development organizations development teams are quite fragmented. Single stakeholders can only maintain a small part of a large system. As a result product line engineers need to modularize and organize the modeling space regardless of the concrete modeling approach used. There are several strategies for structuring and organizing the modeling space:

2.1. Mirroring the solution space structure

Whenever product lines are modeled for existing software systems, the structure of already available reusable assets can provide a starting point for organizing the modeling space. Models can be created that reflect the structure of the technical solution, e.g., separate variability models for different subsystems of a product line. Similarly, the package structure of a software system or an existing architecture description can serve as an initial structure. The number of different models should be kept small to avoid negative effects on maintainability and consistency. This strategy can be suitable for instance if the responsibilities of developers and architects for certain subsystems are clearly established.

2.2. Decomposing into multiple product lines

On a larger scale complex products are often organized using a multi product line structure (Reiser and Weber, 2006). For example, there may be separate product lines for different target customers, e.g., mobile phone product lines for senior citizens, teenagers, and business people (Jaaksi, 2002). Other examples are complex software-intensive systems such as cars or industrial plants with *system of systems* architectures, which may contain several smaller product lines as part of a larger system. Models have to be defined for each of these product lines and must be kept consistent during domain and application engineering. This strategy often means that different teams create and maintain variability models for the product line they are responsible for.

2.3. Structuring by asset type

Another way of dealing with the scale of product line models is to structure the modeling space based on the asset types in the domain. Separate models can then be created for different types of product line assets. Examples are requirements variability models based on use cases (Halmans and Pohl, 2004), architecture variability models (Dashofy et al., 2001), or models for technical and user documentation (John, 2002). Structuring by asset type allows managing variability in a coherent manner. It is however important to manage the dependencies between the different types of artifacts which can easily cause additional complexity. This strategy works well with orthogonal approaches (Pohl et al., 2005) that suggest using few variability models that are related with possibly many asset models.

2.4. Following the organizational structure

This strategy suggests following the structure of the organization when creating product line models. Different stakeholders