Supporting the Evolution of Product Line Architectures With Variability Model Fragments

Deepak Dhungana Thomas Neumayer Paul Grünbacher Rick Rabiser
Christian Doppler Laboratory for Automated Software Engineering Johannes Kepler University, 4040 Linz, Austria
{dhungana | neumayer | gruenbacher | rabiser}@ase.jku.at

Abstract

Evolution is a permanent challenge in product line engineering. Reusable assets such as software components or documents evolve continuously due to new customer requirements or technology changes. This leads to modifications or extensions of the product line’s variability models describing the reference architecture. Due to the large size of product lines, single stakeholders or teams can only maintain a small part of a system which poses additional challenges for evolution. This paper presents a tool-supported approach for building and maintaining variability models of large-scale product lines. We structure variability models into multiple model fragments of manageable size that can be created and maintained by individual teams. Model fragments can be merged semi-automatically into a variability model. We illustrate the approach with examples from ongoing industry collaboration.

1. Introduction

Many of today’s companies adopt a product line approach to foster reuse and to increase productivity of development [14]. The aim of product line engineering (PLE) is to leverage the reuse of core assets and processes, thereby reducing cost and time to market and increasing system reliability and quality [10]. Product line models describe core assets, their commonalities and variability [3, 8]. For instance, reference architecture models describe the variability of architectural elements like components, connectors, or interfaces.

Product line evolution is a big challenge in research and practice. Developing and maintaining a software product line requires dealing with different types of changes caused by evolving customers’ needs, technology, or market developments. Despite its importance, comparably little work is available on product line evolution (e.g., [1, 9, 13]). Many existing approaches assume a fairly stable product line for the definition of domain and variability models. Through collaboration with our industry partner Siemens VAI, we have learned that such stability cannot be taken for granted. Rather, we have observed the continuous evolution of the product line in both domain and application engineering activities. We thus believe that PLE should treat evolution as the normal case and not as the exception.

When analyzing the development practices and the application engineering process of our industry partner we identified two key issues regarding product line evolution: (i) The complexity of the systems requires their development and maintenance by multiple teams who are typically in charge of particular subsystems only. Knowledge about the variability is thus distributed across multiple stakeholders and teams [2]. (ii) Different parts of the system evolve at different speeds further contributing to the complexity of evolution [2, 7]. The main contribution of this paper is a multi-team modeling approach for decentralized creation and maintenance of product line variability models. It allows different teams to work with variability models for their subsystems independently and provides tool support to merge the model fragments created by different teams.

2. Approach

Model fragments describing assets and their variability of selected parts of the product line represent the units of evolution in our approach (Figure 1). In this paper, we present the edit-merge-freeze paradigm. (1) Engineers in different teams edit variability model fragments; (2) these fragments are merged to one variability model at any time required. (3) The merged model is used for product derivation and must not be changed (freeze). Other aspects of our approach such as meta-model evolution and round trip between models and architecture will be discussed in future work.
A variability model fragment represents a variability model of an arbitrary subset of the product line (e.g., a set of features, a subsystem, or cross-cutting functionality). It represents the basic unit of evolution and is created and maintained by individual teams responsible for this subset of the system. We use an orthogonal, decision-oriented approach to variability modeling influenced by the work of Schmid and John [12]. Variability model fragments contain Assets and Decisions as the two key modeling elements [3-5]. Assets (e.g., components, properties, resources etc.) are elements of the architecture model. Decisions represent points which require user input for the selection of a concrete modeling element (e.g., a variant of a component) during product derivation. They allow variability management across different types of product line assets. A model fragment may contain name references to model elements in other model fragments to ensure inter-team (inter-model) synchronization.

A variability model is created by merging a set of model fragments at any time required. Unlike model fragments it can be used in product derivation [11] as all name references contained in the constituent fragments are resolved during merging. The resulting model is frozen and must not be changed.

A merge log is used to establish traceability links between the model fragments and the variability model. Model fragment owners can use the log to revise their fragments based on the applied conflict resolution actions and to expedite future merge processes.

2.1 Defining Variability Model Fragments

Different teams work on variability model fragments describing the parts of the system they know best. Development teams demand a high degree of flexibility for creating and evolving variability model fragments to avoid being constrained by other teams, e.g., when planning new features. It is thus desirable to create and evolve the model fragments without explicit coordination to give teams the leeway necessary for product line evolution.

Working with multiple variability model fragments, however, requires a mechanism allowing users to refer to model elements in other fragments. We support a “lazy consistency” approach that is based on the idea of “loose references”. When creating reference elements, it is not necessary for the user to know how the references will be resolved. The explicit location or the exact names of the referenced elements are not needed during modeling. For example, a decision dbRequired in model fragment 2 (Figure 2) is a reference decision defined in some other model fragment. The person creating this model doesn’t need to know that the real name dbSupport of the element she is referring to. During merging dbRequired is replaced with dbSupport contained in model fragment 1 to resolve this ambiguity.

We adopt well-known concepts from object-oriented programming languages to define the visibility of model elements. Similar to elements in classes a developer can specify public elements of a fragment to make them visible outside the model. Model elements are defined as private elements if they are not relevant to other parts of the system and must not be known outside for variability modeling. Reference elements are introduced in a model fragment, whenever relationships to elements from other model fragments are defined. This allows other subsystems to use the referenced elements as part of their own variability model. This is for instance necessary when specifying product composition rules between elements. For example, model fragment 1 in Figure 2 contains the reference decision archive that is used to define the dependency between dbSupport and archive. Ambiguities introduced by the additional leeway are resolved during subsequent model merging.
2.2 Merging Model Fragments

Merging support is needed to create a variability model containing all the elements of constituent model fragments by resolving references. This is necessary to allow product derivation. Reference elements are bound to real elements while the source model fragments remain unchanged. Figure 3 depicts the result of merging the two model fragments depicted in Figure 2.

In ideal cases, all reference elements match a public element in other models. However, as the model fragments are initially created without explicit coordination and based on “loose references” diverse conflicts can occur during merging.

![Figure 3: Result of merging the two variability model fragments from Figure 2.](image)

**Name mismatches:** The name of a reference might not match the name of its definition. Such cases are difficult to resolve fully automatically and we rely on a human expert during merging to confirm the semantic equality of the element names used in the models. However, our tools also adopt domain-specific glossaries defining synonyms for names to ease merging.

**Multiple definitions/elements with same name:** Different model fragments may define the variability of a common part of a system. This can for example happen when shared components are used by more than one subsystem, and several subsystem owners decide to model the shared components’ variability as a part of their subsystem. Our algorithm detects all element instances (based on naming conventions, types of element, and relationships among elements) and includes only one instance in the merged model. For example, whenever a component with the same name is contained in more than one of the constituent models, the user either decides to (i) rename one of the components before merging; or (ii) to include the component only once in the merged model.

**Missing resolutions of references:** It is also possible that certain elements are not modeled but only referred to using reference elements, e.g., when no team feels responsible for a certain part of the system or if incomplete model fragments are merged. As a result several model fragments might define reference model elements for which no real model element exists. Again, user intervention is required to resolve the problem. The user selects a binding element from a list of suggested candidate elements. If no binding element is available, the resulting variability model will still contain unresolved references and can not directly be used for product derivation.

Whenever model elements are renamed (in case of name conflicts), deleted or dropped (in case of multiple occurrences), element attributes, constraints and conditions also need to be updated accordingly. E.g., the condition “visible if dbRequired==true” in model fragment 2 of Figure 2, was automatically changed to “visible if dbSupport==true” during the merge process. This is because the reference element dbRequired was mapped to dbSupport. Figure 4 shows the model merger tool included in our variability modelling tool DecisionKing [4-6]. The left pane depicts the models to be merged (second model is hidden in this screenshot due to space limitations). Elements with conflicts are marked with a red indicator. For better usability these conflicts are separately listed in the conflict list shown in the middle pane ordered by the type of conflict.

![Figure 4: Model merger tool. Users are presented with a list of auto-generated conflict resolution suggestions.](image)

We record the applied changes and reference bindings in a merge log that maintains traceability information (cf. Figure 1). Change actions performed automatically by the merging tool are also recorded (e.g., renames or reference-mappings). This merge log enables three important features:

**Forward and backward traceability:** It serves as a bridge between model fragments and the variability model to support forward traceability (“how is my model fragment used in the variability model?”) and backward traceability (“from which model fragment does a certain element come from?”).
Feedback to model fragment owners: The original model fragments remain unchanged after merging. However, model fragment owners are informed about conflict resolution changes such as deleting elements and resolving references. This helps avoiding that teams slowly disconnect from each other over time. They may decide to revise their model fragments based on this feedback. This helps teams to converge and agree on definitions in the model fragments.

Repeatability of merging: In case of frequent changes to the fragments and a high number of fragments repeating the merging process each time from scratch can be tedious. Whenever the merge process has to be repeated after a change to a model fragment, the merge log is used to replay the previously taken change actions with minimal user intervention.

3. Conclusions and Future Work

In this paper, we introduced the Edit-Merge-Freeze paradigm, useful for maintaining large product line models. An approach based on this paradigm allows working with comparably small models (fragments) which reduces complexity of their creation, maintenance, and evolution. Changes can be made locally in specific model fragments. A model of the complete system is created by merging the fragments, as required. This also supports the evolution of different subsystems at different speeds.

We have been experiencing the evolution of Siemens VAI’s product line over the last two years. We were able to validate our approach by iteratively developing and testing evolution capabilities using real-world product line variability models. Our tools have been used by engineers at Siemens VAI to create and evolve variability models. It became apparent early on that working with a single variability model is inadequate to support evolution in the multi-team development environment at Siemens VAI which led us to develop a multi-team approach.

Our future work will continue in the following directions: We will develop guidelines that help product line engineers to decide in which model fragment a subsystem’s variability should be modeled. Furthermore, we will investigate ways of handling the evolution of variability in product lines. We will also explore visualization support for different aspects of our approach.

References


