Configuration Management for Software Product Lines

Roland Laqua and Peter Knauber

Fraunhofer Institute for Experimental Software Engineering (IESE)
Sauerwiesen 6
D-67661 Kaiserslautern, Germany
+49 6301 707 161
{laqua, knauber}@iese.fhg.de

ABSTRACT
This position paper proposes a new approach to support management of the assets used for software product line development. Examples for these assets are the domain (or product line) model, reference architecture, design, and code. New configuration management concepts provide means to create, maintain, and evolve these assets efficiently and consistently, including not only their common parts but product-specific parts as well.

1 Introduction

The basic concept behind software development in product lines is goal-oriented development for and with reuse. Reuse is achieved through synergistic effects during development and maintenance of common system parts. In order to plan and develop these common parts to be usable by future members of the product line, it is necessary to consider the main variabilities among the future products during modeling, architecting, design, and implementation. The commonalities are represented together with the main variabilities in generic assets.

There are two reasons for this separate treatment:

- First, tools to support the manipulation of generic models including model instantiation[1] are basically not available. Thus, in traditional product line approaches, tools for single system development are adapted or extended in such a way that they provide some support for genericity, at least the respective notation.
- Second, keeping them separate from each other allows for the parallel development and maintenance of product line members.

Two general kinds of processes have to be considered for generic product line assets:

- feed forward processes to derive them and to add new or change existing functionality and
- feed back processes to (re-)integrate changes or additions to product-specific assets into the common core in order to reuse them in other products as well.

Feed back processes, especially, are difficult and error prone because traditionally, product-specifics are developed (and typically also stored) separately from the generic assets representing the common core. Thus, it is difficult to decide if and which product-specific aspects should be integrated into the core and it is even more difficult to perform the integration.

The lack of appropriate tools makes integration of product specifics into the common core extremely difficult: there is basically no automated support for checking constraints on existing models if these constraints apply to model variations, or to even represent them explicitly.

Even traditional configuration management tools do not provide enough support for these particular problems (e.g., constraint checking on system variants): They allow representation and checking of constraints but do not support product variants (e.g., alternatives or options).

The position we present in this paper can be briefly summarized as follows:

> In order to support management of product line assets appropriately, the definition of the way generic asset data are structured has to be reflected in the supporting configuration management system. This system has to be set-oriented, attribute-oriented, unification-oriented, and structure-oriented.

This position will be explained in the remaining sections of this paper. Section 2 presents the basic assets of a product line infrastructure, in section 3 the configuration management elements on which these product line assets have to be mapped are explained. Problems of traditional configuration management during maintenance and evolution of a product line are shown in section 4. Finally, the proposed configuration management prerequisites for the maintenance of a product line infrastructure are presented in section 5.

2 Software Product Line Assets

The traditional software product line (SPL) infrastructure consists of

- assets representing the product line (which are the smallest units from a configuration management point of view),
- the relations between the assets, which are captured by the product line configurations.

1.The term “instantiation” here refers to hiding model parts that do not belonging to a certain product and show the parts belong only to this product while checking existing constraints.

Product line assets can be considered as intermediate software products that are needed to produce applications in a certain domain. They can be subdivided into domain assets that capture the knowledge of the business domain, and system assets representing the information specific for certain systems of the product line, which are needed to derive (e.g., parametrize, generate, instrument) the assets for these specific systems from the (generic) domain assets.

Domain assets can be categorized into three groups: *common assets* (holding information that is identical for all members of the product line), *generic assets* (for capturing information about variant parts of the software product line in one asset), and *configuration assets* (representing the information needed to configure systems in the product line).

- **Common assets** only hold information that is common for all members of the product line and contain no information specific to a certain system. These assets should not vary because they are used in the same way by all members of the product line.
  Examples are database access libraries, which very often can not be changed by the developers, or trace and log services, which can be shared by most systems of a product line.
- **Generic assets** capture information about different variants of a product line asset in one common asset in order to increase its reusability and adaptability for different systems of a product line. The description of the different variants strongly depends on the SPL method used, very often it is achieved by means of the introduction of variation points via conditional blocks or by feature modeling methods (see [1],[2] pp.82).
  An example can be seen in [3], [5] p.8. There the optional and alternative sub-components are decorated with conditional elements that are used to decide whether or not the sub-component will be part of the specific product.
- **Configuration assets** contain the information on how to derive a specific asset from a generic one.
  An example is the decision model that describes the relation of possible decisions ([4]) in the business domain and variation points of a certain asset and how they can be resolved.

System assets as mentioned above represent the information specific to a certain system of the product line and the knowledge needed to build a certain system of the product line.

- **Specific assets** describe information that is specific to a certain system or sub-group of systems in the product line. Typically they contain no variabilities and share less common parts with other assets.
  Examples are the description of the compilation parameters for a certain asset or GUI forms, which are very often system specific.
- **Production assets** define a certain system of the product line. They configure the different parts that belong to a system and resolve the decisions that are necessary to derive specific assets from generic ones.
  Examples are tuples of decisions that resolve variation points in generic assets or makefiles that contain the dependencies between assets belonging to a certain system.

Product Line System
A product line system defines all the domain and system assets necessary to build a specific application from the product line infrastructure. This means that all decision tuples that are needed to resolve the generic assets together with the appropriate decision models will be described as part of a product line system. Moreover, all application specific assets used are defined there.

3 Configuration Management Elements

The main objective of configuration management (CM) in SPL is to support maintenance of the product line infrastructure. This means to support management of the life cycle of the product line assets, their different customer specific variants and revisions, their configuration of the assets in specific systems, and their change over time.

In order to meet these requirements, CM systems must be able to map SPL elements onto their own structures and to support the different SPL processes like derivation of specific assets and reintegration of changes. The basic elements of CM systems to implement these requirements are *items* (which are the elementary units), *configurations* (which define the relations between items), *revisions* (for describing the change of items and configurations over time), and *variants* (which define different shapes of the same item or configuration).

Items
Items are elementary units of a software product that must be uniquely identified. They can be instantiated for different versions (see “Version” on page 2). This definition is open to any kind of software product because no type restriction is made.

Items can be categorized in two groups:

- **plain items** that are not derived from any other items (but nevertheless can depend on others) and
- **derived items** that are derived from one or more other items (which again can be plain or derived).

This distinction is needed because the versions of derived items also need the version information about the plain items they stem from.

Version
A version is an instance of an item. Two instances of the same item that are developed in parallel are related as variants, whereas a version that follows by a change of another version of the same item is related to this version as a revision. The variant relation defines alternative forms, whereas a revision expresses the change of an item over time.

Configuration
A configuration defines a composition of items and possibly further configurations. Configurations have to obey the following rules: First, a configuration can only be assigned to one version of a given item, second, a version of an item

may be a member of several configurations, third, a configuration can be recursive, and fourth, a configuration version can be assigned to another configuration.

Mapping SPL to CM elements

The mapping of SPL elements to CM elements creates a certain net of plain and derived items of generic, common, specific, production, and configuration assets. The possible relations between SPL and CM elements, as well as the source assets they are derived from, are depicted in Table 1.

SPL Asset	Corresponding CM Asset	Source of Derivation
Generic Asset	Plain Item	Plain Item
	Derived Item	Generic Asset + Configuration Asset + Production Asset
Common Asset	Plain Item	
	Derived Item	Common Asset + Production Asset
Configuration Asset	Plain Item	
Specific Asset	Plain Item	
	Derived Item	Specific Asset + Production Asset
		Generic Asset + Configuration Asset + Production Asset
Production Asset	Plain Item	
System	Configuration	Subset of Generic Assets + Subset of Common Assets + Subset of Specific Assets + Production Assets

Table 1: SPL - CM element map

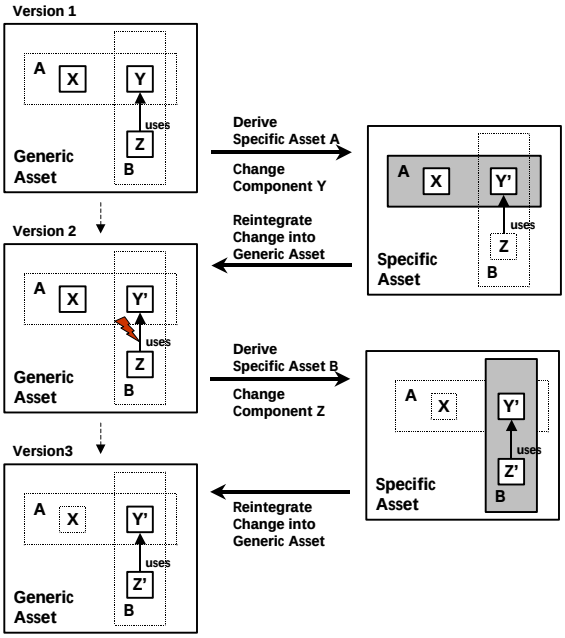

Figure 1: Consistency Problem

4 Problems of Traditional CM for SPL

The traditional approach suffers from certain problems that arise in the different activities of the change processes and that are related to the fact that the product line assets are treated as self-contained items in a CM system without considering the inner structure of an asset. Typical problems are the *reconstruction problem*, the *change propagation*, and the *consistency problem* (which are related to the correctness of the product line and which emerge during the change process), and the *set problem* (which describes difficulties in treating a set of systems instead of a single system).

Consistency Problem: The reintegration of changes made in specific assets can lead to inconsistencies in another specific asset that is derived from the same generic asset. Therefore, the reintegration raises the need to adapt parts of the other specific asset.

This happens, for example, if a certain part of a specific asset B (figure 1) uses a common part that is shared by a specific asset A and that was changed in the specific asset A. The derivation of B from version 2 of the generic asset would then lead to an inconsistent specific product B. This implies that before the generic asset can be used for all specific assets that can be derived from it, all change effects have to be determined and the changes have to be integrated for reaching consistency again.

Reconstruction Problem: The parallel change of specific assets derived from the same generic asset can lead to specific assets that are not reconstructible. Figure 2 shows an example, where two specific assets A and B are changed at the same time and these changes are then reintegrated into the original generic asset at different points in time (e.g., A before B). This implies that the changed specific asset B can not be reconstructed from any of the three generic asset versions because in version 1 and version 2 the part Z', and

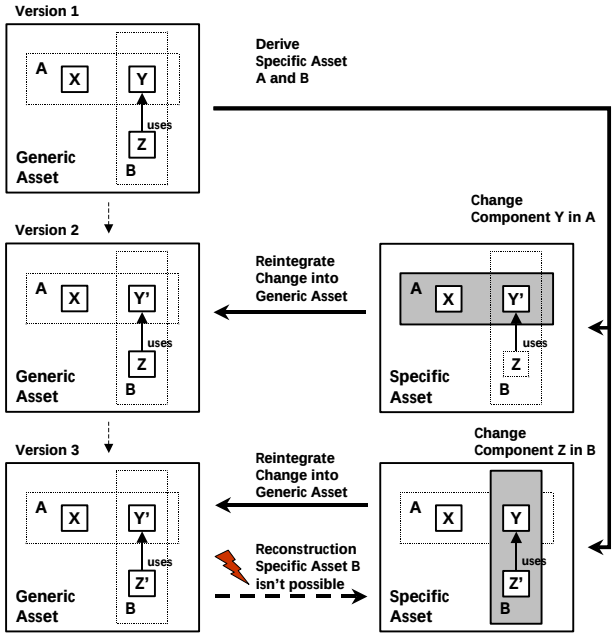

Figure 2: Reconstruction Problem

in version 3 the part Y are missing.

Set Problem: The simultaneous change of common parts of specific asset variants derived from the same generic asset is impossible or hard to achieve. Changes in the traditional CM only allow working with one instance of the product line in parallel. This is because each instance is treated as a certain variant of the product line instead of handling all its variants as a set.
Furthermore, the identification and selection of common parts is not supported by traditional CM systems. Figure 3 illustrates the problem and a possible solution through a respective specification of the affected asset parts with an example.

Change Propagation Problem: Traditional CM systems are not able to represent revisions and variants of a product line in one common model. Thus, there is no possibility to propagate changes for more than one variant at once. This is depicted in figure 4, where a change in a specific asset derived by generic asset version 1.0 has to be reintegrated twice, once in the generic asset version 1.1 and then in the generic asset version 2.1 of the generic asset variant version 2.0.

5 Proposed CM Concept for an SPL Infrastructure

In order to overcome these problems, we propose a CM concept that has the following four major properties:

Set-orientation: The CM concept supports manipulating consistent sets of items and configurations. For this it provides operations to identify and select them according to their properties as well as to manipulate them at once.

Attribute-orientation: The CM concept supports

- identification and selection of plain, derived and composed items that are tagged with attributes,
- propagation of attributes among the items, and
- design of relations among the items in order to enable attribute propagation.

Unification-orientation: The CM concept supports

- unification of an attribute model for all items of the product line and
- identification and selection schemes for assets.

In the most CM systems a strong identification scheme comes along with a weak selection scheme or vice versa. For example, the selection of variants can be realized by the C preprocessor that provides a strong variant identification mechanism via arithmetic expressions. On the other hand, the selection of versions through the conjunction of attributes is quite complex using that tool. In order to overcome this problem different steps are necessary:

- Firstly, items (configurations) and SPL assets (SPL Infrastructure) are unified. This means that they are the same entity inside of the repository (see Figure 5, Figure 6 melting). Thus, the structural model of an SPL asset can be used to build up CM approaches for versioning, change propagation and a unified selection and identification scheme using of the repository meta model.
- Secondly, items and configurations in the repository are mapped on a common feature logic attribute model. Thus, the selection and identification scheme of items and configurations can now rely on the same expression model for both.
- Thirdly, the items (configurations) are modeled and interpreted as features themselves. This means that additional features decorating items (configurations) are treated in the same way as the items (configuration). This way, the identification and selection scheme of items and their attributes can be unified in one expression.

Structure-orientation: The CM concept supports the definition of optional and alternative assets and asset parts for product lines.

Mapping of product line assets to CM items
Each product line asset has to have a dual nature (figure 5): on the one hand it is handled as a CM item, and, on the other hand, it fulfils its purpose as a product line asset.

To reach this goal the data structure of each asset has to be modeled on the data structure level of the CM repository. Thus, it will be possible to decorate each entity (figure 5) of the data structure with features like the ones in feature logic ([5]). There, a feature is not only the decoration of an entity but also the entity itself. Even the changes of an entity can be

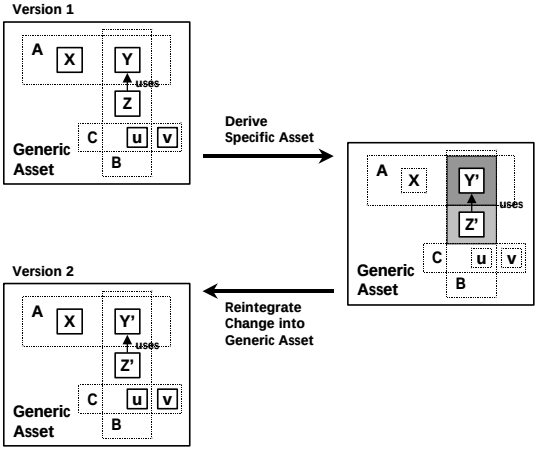

Figure 3: Set Problem

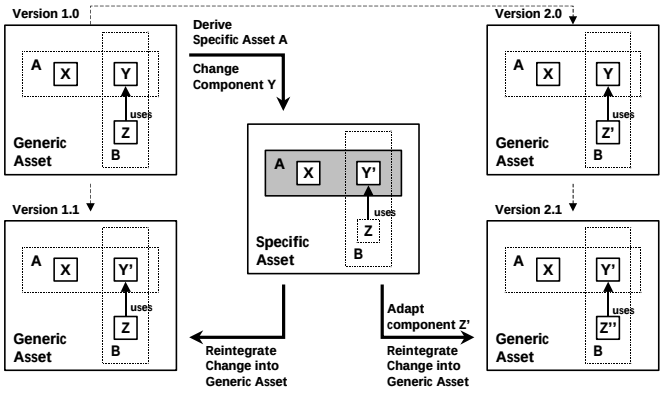

Figure 4: Change Propagation Problem

seen as a feature, enabling the unification of variants and revisions in one common model. Thus, product line assets and CM items can be modeled and treated on the same level.

Advantages of the proposed concept
The proposed CM concept solves the problems listed in section 4:

- **Consistency Problem**: The set-orientation and the structure-orientation allow to define sets of specific assets. That allows to change dependent parts of different specific assets derived from the same generic asset in parallel.
- **Reconstruction Problem**: The attribute-orientation allows to unify parts of different revisions that were developed in parallel into one common configuration item. Therefore, the reconstruction of certain specific assets created during development is always possible.
- **Set Problem**: The set-orientation allows to define subsets of arbitrary parts of specific assets derived from generic assets. Changes to these sets affect all selected assets at the same time.
- **Change Propagation Problem**: Set and attribute-orientation enable to group parts of different variants of specific assets that were developed in parallel into a common set and to change them at one time. This makes multiple reintegration superfluous.

The realization of such an SPL infrastructure requires anchoring the product line asset and configuration models in a version-set-oriented repository ([6]). This is necessary to interpret a certain CM item as a set of different variants depending on the properties on the data structure level, as must be done for generic product line assets. Additionally, it is necessary to explicitly express the relations among different assets. This will be done by the feature logic approach (see [5]) based on the product line assets' data structure. This allows to design the dependencies and attributation on a finer grained level than with traditional CM concepts.

6 Conclusion

In this paper we present a new concept for configuration management to support management of software product line assets. Typical problems of traditional approaches in a product line context are described and their solution using the new concept is sketched. Future work will cover the refinement and, later on, the implementation of the concept using existing tools (if possible).

REFERENCES

[1] K.C. Kang, S.G. Cohen, W.E. Novak, A.S. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study, Tech. Report CMU/SEI-90-TR-21, Software Engineering Institute (SEI), November 1990

[2] K. Czarnecki, U.W. Eisenecker, Generative Programming, Addison-Wesley, May 2000

[3] D.M. Weiss, Defining Families: The Commonality Analysis, Lucent Technologies Bell Laboratories, 1997

[4] C. Atkinson, J. Bayer and D. Muthig, Component-Based Product Line Development: The KobrA Approach, 1st International Software Product Line Conference, Pittsburgh, August 2000.

[5] A. Zeller, Configuration Management with Version Sets, A Unified Software Versioning Model and its Applications, PhD thesis, http://www.cs.tu-bs.de/softech/papers/zeller-phd

[6] C. Reichenberger, Konzepte und Verfahren für die Software-Versionsverwaltung, Universitätsverlag Rudolf Trauner, Linz 1994

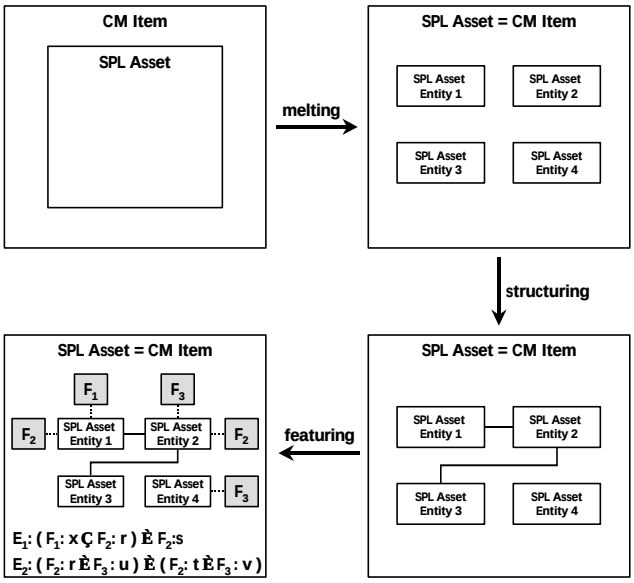

Figure 5: Feature Set Items

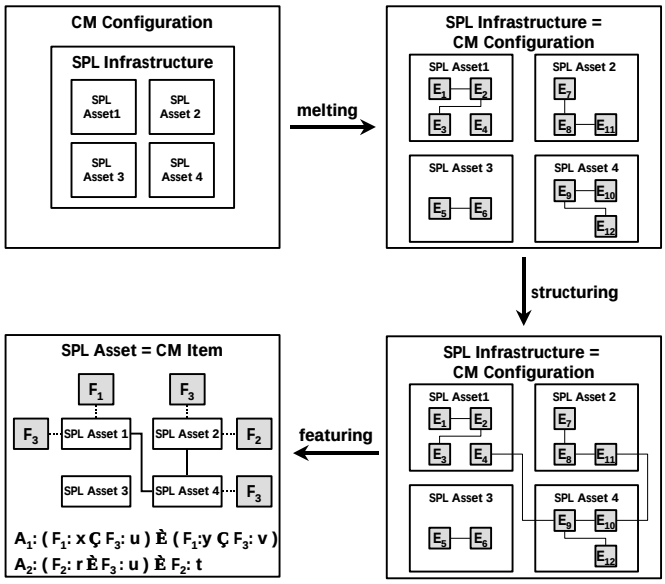

Figure 6: Feature Set Configurations