Expressing Feature-Based Variability in Structural Models

Iris Groher[1], Markus Voelter[2]
[1] *Siemens AG, CT SE 2, Munich, Germany*
[2]*Independent Consultant, Goeppingen, Germany*
iris.groher.ext@siemens.com, voelter@acm.org

Abstract

Software product line engineering aims at reducing development time, effort, cost, and complexity by taking advantage of the commonality within a portfolio of similar products. The effectiveness of a software product line approach directly depends on how well feature variability within the portfolio is implemented and managed throughout the development lifecycle, from early analysis through maintenance and evolution. This paper presents a tool-supported approach that improves variability management and tracing by providing means to express feature-based variability on model level. Features are separated in models and automatically composed. The approach supports both positive variability, i.e. adding optional parts to a model, as well as negative variability, i.e. removing parts from a model. Tools are provided that implement the presented concepts. The approach is illustrated with a case study of a home automation system.

1 Introduction and Motivation

The effectiveness of a software product line approach directly depends on how well feature variability within the portfolio is managed from early analysis to implementation and through maintenance and evolution [1][2]. Commonalities, as well as the flexibility to adapt to different product requirements are captured in core assets. Those reusable assets are created during domain engineering. During application engineering, products are either automatically or manually assembled, using the assets created during the domain engineering process and completed with product-specific artifacts. Products usually differ by the set of features they include in order to fulfill customer requirements. A feature is an increment in functionality provided by one or more members of a product line [3].

Variability management is the activity concerned with identifying, designing, implementing, and tracing flexibility in software product lines (SPLs). Variability of features often has widespread impact on multiple artifacts in multiple lifecycle stages, making it a predominant engineering challenge in software product line engineering (SPLE).

Our approach integrates model-driven software development [4] and product line engineering by providing means for expressing variability on model level. We argue that due to the fact that models are more abstract and hence less detailed than code, variability on model-level is inherently less scattered and therefore simpler to manage. Variability can be described more concisely since in addition to the traditional mechanisms (e.g. patterns, frameworks, polymorphism), variability can be described on the more abstract level of models. This paper focuses on concepts and tools that support the expression of feature-based variability in structural models and hence the selective adaptation of models.

Tools are provided that support both positive and negative variability on model level. In the case of negative variability, models are tailored based on the absence of features defined in a configuration model. This means that if a certain feature is not part of the configuration, the model elements implementing this feature are removed from the model. In the case of positive variability, optional parts are defined in a separate model and only added to the core model if a certain feature is present in a configuration. This is what we call *model weaving*, an application of aspect-orientation on model level [14][15].

The tools and concepts presented in this paper are important ingredients of what we call *Aspect-Oriented Model-Driven Product Line Engineering (AO-MD-PLE).* While this paper concentrates of the techniques to define structural variants of models, another paper [8] focuses on the creation of variants of model-to-model transformations and code generators.

The overall approach is illustrated in [7]. Here is a short summary of the core idea:

AO-MD-PLE uses models to describe product lines. Variants are defined on model-level. Transformations generate running applications. AO techniques are used to help define the variants in the models as well as in the transformers and generators.

Figure 1 illustrates the key parts and the corresponding models of AO-MD-PLE. Domain requirements are captured in a problem space meta model. Based on product requirements, a problem space model is created that is an instance of the problem space meta model. A formal mapping is defined between the problem space meta model and the solution space meta model which allows for the automatic transformation of the problem space model into the solution space model. Based on this model a product is automatically generated using the core assets created during domain engineering.

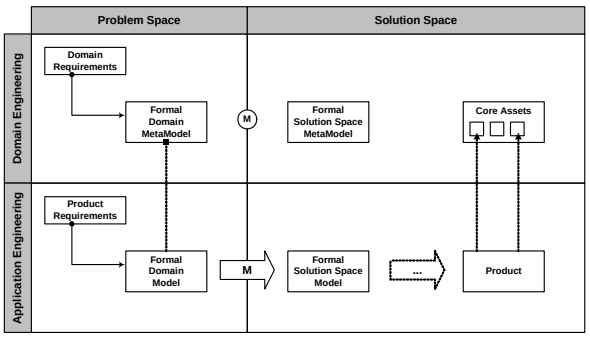

Figure 1: Aspect-Oriented Model-Driven Product Line Engineering

The remainder of the paper is organized as follows: Section 2 demonstrates how to implement feature-based variability in structural models and tools that realize the presented concepts. Section 3 illustrates the case study and how the introduced concepts were applied there. Related work is discussed in Section 4, while Section 5 summarizes the paper and provides an outlook on future work.

2 Expressing Variability in Structural Models

2.1 Model-Driven Software Development

Model-driven software development (MDSD) [4] improves the way software is developed by capturing key features of the system in models which are developed and refined as the system is created. During the system's lifecycle, models are combined and transformed between different levels of abstraction. In order to be processable by tools, models have to be formal. This means that every model is an instance of a well-defined meta model representing the abstract syntax of the model.

A Domain Specific Language (DSL) [4] is a formalism for building models: It encompasses a meta model as well as a definition of a concrete syntax that is used to represent models. The concrete syntax can be textual, graphical or using other means such as tables, or trees.

2.2 Kinds of Variability in Models

We distinguish between two kinds of variability: structural and non-structural. Structural variability is described using creative construction DSLs, whereas non-structural variability can be described using configuration languages. Figure 2 illustrates the spectrum of languages commonly used for expressing and binding variability.

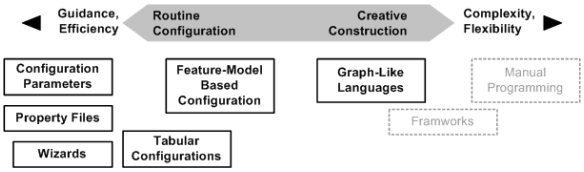

Figure 2: Expressive power of DSLs

Figure 3 shows the meta model of a creative construction DSL. It is a meta model that can be used for creatively constructing data structures. Any number of models can be defined, by instantiating and composing meta model elements. Figure 4 presents two example models a) and b) that can be constructed with a DSL (using the familiar concrete syntax of UML) that implements the meta model in Figure 3.

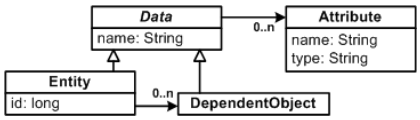

Figure 3: Creative construction meta model

Figure 5 shows a feature model of a stack using the notation defined in [9]. The feature model expresses a certain configuration space, i.e. the model is an expression of configurative variability. A specific configuration is described by selecting a valid subset of those features according to the constraints expressed by the feature model.

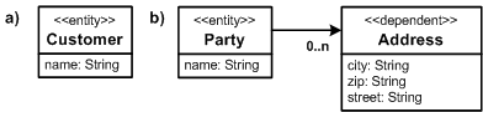

Figure 4: Example models

To align feature modeling with general MDSD terminology, it is useful to consider the feature model a meta model and the concrete configurations models.

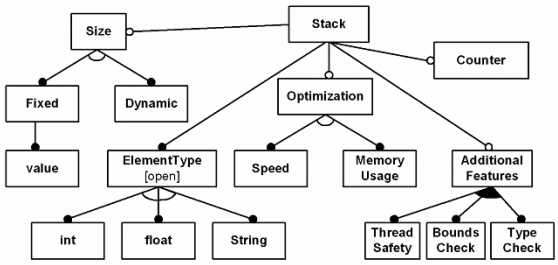

Figure 5: Feature model of a stack

We have shown that there are two fundamentally different kinds of variability, and consequently, two different kinds of DSLs: creative construction DSLs and configuration DSLs. Models built with creative

construction DSLs (we call those structural models) often have to be adapted based on a product configuration. In other words, we want to use a configuration model to define variants of a structural model. This is especially useful in the context of software product line engineering.

As an example consider a creative construction DSL for home automation systems. Such a DSL allows to creatively connect devices to rooms and floors. Such a model can be adapted using a configuration DSL. Additional features such as security can change the creative construction model by for example adding motion detectors and fire sensors.

The next sections show how this can be achieved using our approach and how available tools implement the presented concepts.

2.3 Implementing Positive Variability in Structural Models

Positive variability starts with a minimal core and selectively adds additional parts based on the presence or absence of features in the configuration models (see Figure 6).

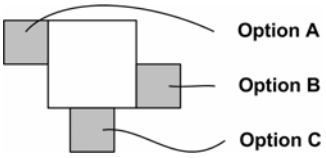

Figure 6: Positive variability

Model weaving assists in the composition of different separated models into a consistent whole. It allows to capture variable parts of models in aspect models and weave them into a base model. Consequently, the base model is minimal in that it only contains elements common to all products. Product-specific parts are added when needed.

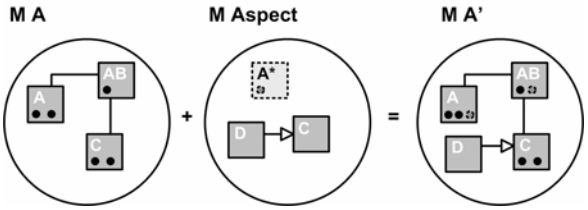

Figure 7: Model weaving

Figure 7 shows how model weaving works. A given base model (M A) and an aspect model (M Aspect) are woven. The aspect model consists of pointcut definitions that capture the points where in the base model the additional model elements should be added. The aspect model also contains these additional model elements (the "advice", in AO terms). After the weaving process a result model (M A') is created that contains the original base model elements plus the aspect elements added at the appropriate points.

This technique allows for a clear separation of optional model parts and supports automatic composition to create a complete model. We developed a tool called XWeave [10] that implements the presented concepts. We will give more details on the tool in Section 2.5 and present how it was used to implement features in the Smart Home case study.

2.4 Implementing Negative Variability in Structural Models

Negative variability selectively takes away parts of a creative construction model based on the presence or absence of features in the configuration models (see Figure 8).

This technique is fundamentally different to the technique introduced in the previous section. When using negative variability to implement feature-based variability in structural models, one has to build the "overall" model manually and connect elements to certain features in a configuration model.

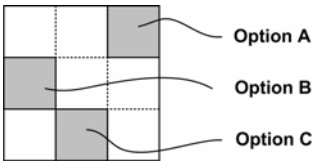

Figure 8: Negative variability

Figure 9 illustrates how negative variability on model level works. It shows a simple feature model of a party (in the sense of participant, e.g. in a contract). A party can either have an international phone number or a local phone number and is variable in how it is made persistent. There are two optional features, namely the possibility of adding state information and multiple addresses. Features of the configuration model are linked to elements of the structural model below. In this case, attributes and relationships between classes are connected to the features. Those elements are only present in the model iff the corresponding features are part of a configuration.

2.5 Tool Support

The tools we built are based on Eclipse [11] as a tool platform, Ecore [12] as the basis for modeling and openArchitectureWare (oAW) [13] as the tool for model processing.

The tool for implementing positive variability is called XWeave[1] [10]. It is a model weaver that can

[1] XWeave and XVar can be downloaded from

weave models that are either instances of Ecore (these are called meta models) or instances of these models (these are called models). XWeave takes a base model as well as one or more aspect models as input and weaves the content of the aspect model into the base model. There are two ways of specifying pointcuts: name matching and explicit pointcut expressions. Name matching means that if a model element in the aspect model has a corresponding element in the base model (both name and type have to be equal), the elements are combined. Pointcuts can also be defined using a dedicated expression language. Expressions can select one or more elements of the base model and are defined in a separate expression file. Expression have a name and can be referenced by this name. It is possible to use wildcards within pointcut expressions to select several join points in the model with only one declarative statement ("quantification", in AO terms). The language for building those pointcut expressions is oAW's extension language [13], a variation of OCL.

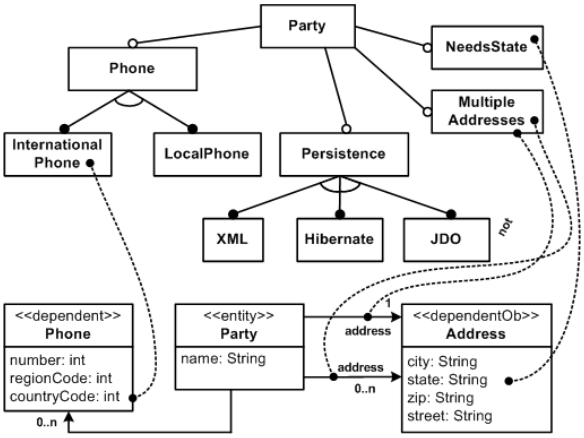

Figure 9: Negative variability

Figure 10 illustrates how XWeave can be linked to configuration models. Aspects that implement optional parts of structural models are linked to features defined in the configuration model. Based on a selection of features, the corresponding model aspects are woven to the base model.

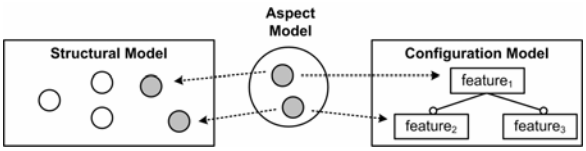

Figure 10: Linking positive variability to configuration models

The tool for implementing negative variability is called XVar[1]. It tailors either models or meta models.

[1] http://www.eclipse.org/gmt/oaw/

Figure 11 illustrates how the tool is linked to configuration models. A dependency model captures the relationships between model elements and features. According to a selection of features, the structural model is tailored to only contain the model elements needed for the respective configuration.

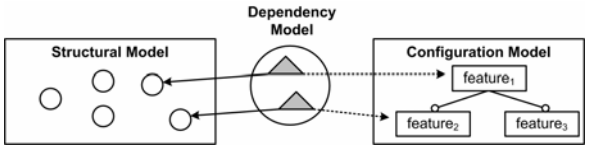

Figure 11: Linking negative variability to configuration models

Concrete examples on how to use the tools are provided in the next section that illustrates the case study.

3 Home Automation Case Study

The case study to illustrate our approach is a home automation system (see also [1]), called *Smart Home*. In homes you will find a wide range of electrical and electronic devices such as lights, thermostats, electric blinds as well as fire and smoke detection sensors. Smart Home connects those devices and enables inhabitants to monitor and control them from a common UI. The home network also allows the devices to coordinate their behavior in order to fulfill complex tasks without human intervention.

Sensors are devices that measure physical properties of the environment and make them available to Smart Home. Controllers activate devices whose state can be monitored and changed. Varying types of houses, different customer demands, the need for short time-to-market and saving of costs drive the need for a Smart Home product line and are the main causes of variability. Figure 12 shows an example house. It contains one floor, the cellar, and two rooms including several devices.

Figure 12: Example house

In the remainder of this section we will explain how to use the techniques introduced in Section 2 to imple-

ment the Smart Home product line.

3.1 Implementing Features using Positive Variability

This section provides an example of using positive variability to implement an optional feature of the home automation product line. The *automaticWindows* feature automatically opens the windows if the temperature is a room is above a certain threshold and closes them again if the temperature is below a certain threshold.

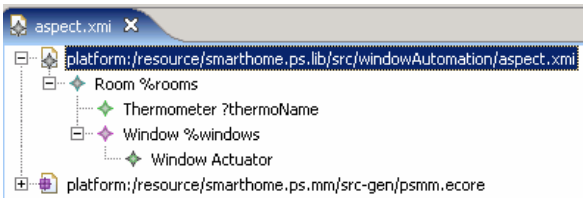

Figure 13: Window automation aspect

In order for this feature to be included in a configuration, the necessary devices have to be woven into the building model. Figure 13 shows the aspect model that is responsible for weaving the respective features into the example house shown in Figure 12. The pointcut expressions used in the aspect model are the following:

```
rooms (Building this) :
        floors.rooms.select (e|e.windows.size > 0);
windows (Building this) :
        rooms().windows;
thermoName (Thermometer this) :
        ((Room)eContainer).name.toFirstLower() + "Thermometer";
```

`Rooms` returns all rooms that have windows. To all of them a thermometer should be added and `thermoName` is a helper function that creates a sensible name for this thermometer. `Windows` returns all windows of these rooms and a window actuator is added to it.

The resulting (woven) model has a thermometer in each room to measure the current temperature and a window actuator for each window to be able to automatically open the windows.

The dependency between the aspect and the feature is specified in the workflow. If the automaticWindows feature is present in the configuration model, XWeave weaves the content of the aspect model into the house model. The following code snippet shows the respective workflow:

```
<feature exists="automaticWindows">
  <cartridge file="org/openarchitectureware/util/xweave/wf-weave-expr"
        baseModelSlot="houseModel"
        aspectFile= "windowAutomationAspect.xmi"
        expressionFile="windowAutomation::expressions"/>
</feature>
```

3.2 Implementing Features using Negative Variability

This section provides an example of using negative variability to implement an optional feature of Smart Home. The *dimmableLights* feature enables inhabitants to set the light level of lights instead of only turning them on and off.

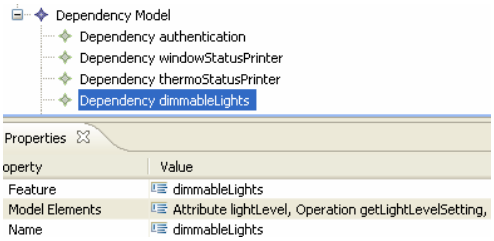

Figure 14: Dependency model

The interface of the light driver contains all operations, including the optional operations to set the light level (see Figure 15). The dependency model shown in Figure 14 manages the relationships between the dimmableLights feature and model elements. According to this dependency model, the base model is tailored in case the feature is not part of the configuration. The interface then only contains operations to turn lights on and off.

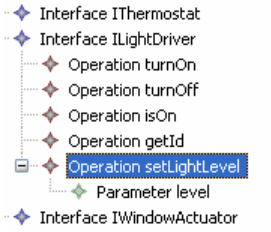

Figure 15: Light driver interface

4 Related Work

AMW, the Atlas Model Weaver [16], is a tool created by INRIA as part of the ATLAS Model Management Architecture. It's primary goal is to establish links between models. In the first phase of working with AMW, a number of links are established between two or more models. This process can be manual or semi-automatic. The result is called weaving model. Based on that model, one can generate model transformations that merge models. AMW is similar to XWeave as both tools can weave or merge models. There are, however, important differences. For example, AMW contains an interactive tool to build the weaving model, whereas XWeave uses name correspondence or pointcut expressions. XWeave integrates well with the rest of the oAW tools. For example, it is compatible with OAW's workflow engine and uses the

oAW expression language.

The C-SAW project [18] is developed by the University of Alabama at Birmingham. It is a general transformation engine for manipulating models based on aspect specifications using ECL (a variant of the Object Constraint Language, OCL). The weaver traverses the model and selects a set of elements to which the aspect should be applied. The advice then modifies the selected element in some way, for example by adding a precondition or changing the element structure somehow. C-SAW has been developed to tackle the challenge of evolving large models in consistent ways. Instead of applying a set of changes manually, one merely writes an aspect that applies the changes to all selected elements in the model. Comparing it to XWeave reveals that C-SAW doesn't weave models (in the sense of merging them) as XWeave does. Rather, it efficiently applies (crosscutting) changes to a collection of elements in a large model.

In [17] structural models are connected with variability models to implement negative variability. A feature model is linked to a UML model via stereotypes and depending on the selected features, the UML model changes. XVar also implements negative variability for structural models but in contrast to [17] it provides a generic EMF based solution. Another important difference is that the links between model elements and features are managed in a separate dependency model in XVar. In [17] the links are managed using stereotypes which requires invasive changes to the model that should be tailored.

5 Summary and Future Work

In this paper we have shown how feature-based variability can be expressed in structural models. When integrating MDSD into software product line development, structural models play an important role. Meta model describe the abstract syntax of DSLs which are used to specify products. Both, meta models and their instances have to be adapted according to the presence or absence of features in configuration models.

We have presented two different approaches of implementing variability on model level: positive and negative variability. Positive variability adds optional parts to a given base, whereas negative variability removes optional parts from a given base. We also presented two different tools that implement those concepts. XWeave uses aspects on model level to weave optional parts into a given base model. XVar uses a dependency model to specify how parts of the model relate to features in the configuration model.

Our approach allows to effectively adapt structural models according to configuration models which is especially useful in the context of software product lines. Features can therefore be expressed on the higher level of models and code generators can remain untouched.

XWeave currently only supports additive weaving. In the future we will also support changing or overriding of model elements in the base model.

6 Acknowledgments

This work is supported by AMPLE Grant IST-033710. The authors would like to thank Christa Schwanninger for her valuable comments on earlier drafts of this paper.

7 References

[1] K. Pohl, G. Böckle, and F. v. d. Linden, *Software Product Line Engineering Foundations, Principles, and Techniques.* Berlin: Springer, 2005.

[2] P. Clements, and L. M. Northrop, *Software Product Lines: Practices and Patterns:* Addison Wesley, 2001.

[3] P. Zave, "FAQ Sheet on Feature Interaction": http://www.research.att.com/~pamela/faq.html

[4] T. Stahl, and M. Voelter, *Model-Driven Software Development:* Wiley & Sons, 2006.

[5] R.E. Filman, T. Elrad, S. Clarke, and M. Aksit, *Aspect-Oriented Software Development.* Amsterdam: Addison-Wesley Longman, 2004.

[6] AOSD website, http://www.aosd.net

[7] M. Voelter and I. Groher, "Product Line Implementation using Aspect-Oriented and Model-Driven Software Development", To appear in *Proceedings of the 11th International Software Product Line Conference (SPLC)*, Kyoto, Japan, September, 2007.

[8] M. Voelter and I. Groher, "Handling Variability in Transformations and Generators", *Submitted for publication*, 2007.

[9] K. Czarnecki and U. W. Eisenecker, *Generative Programming. Methods, Tools, and Applications.* Amsterdam: Addison-Wesley Longman, 2000.

[10] I. Groher and M. Voelter, "XWeave – Models and Aspects in Concert", In *Proceedings of the 10th Workshop on AO Modeling*, Vancouver, Canada, March, 2007.

[11] Eclipse Foundation website, http://eclipse.org

[12] Eclipse Modeling Framework (EMF) website, http://eclipse.org/emf

[13] openArchitectureWare (oAW) website, http://www.eclipse.org/gmt/oaw/

[14] Aspect-Oriented Modelling, http://aspect-modeling.org/

[15] S. Clarke and E. Baniassad, *Aspect-Oriented Analysis and Design. The Theme Aproach.* Addison-Wesley, 2005.

[16] Atlas Model Weaver (AMW) website, http://www.eclipse.org/gmt/amw/

[17] K. Czarnecki and M. Antkiewicz, "Mapping features to models: A template approach based on superimposed variants", *GPCE 2005*, Tallinn, Estonia, September, 2005.

[18] C-SAW project website, http://www.cis.uab.edu/gray/Research/C-SAW/