Managing Variabilities with Generative Approaches

Iris Groher
Siemens AG
Otto-Hahn-Ring 6
81739 Munich, Germany

groher@informatik.tu-darmstadt.de

A software product line consists of a family of software systems that have some common functionality and some variable functionality. Clements and Northrop [1] define a software product line as "a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets".

Feature models play a central role within product line development as they allow a uniform representation of the variabilities and commonalities of the potential products of the product line. These concepts were originally introduced by the Feature-Oriented Domain Analysis (FODA) method [2] which uses features that are organized into a feature tree. Feature models can be used to model the problem domain within the product line development. A concrete feature set then represents the requirements for one concrete product. Feature sets can be chosen by a customer in order to configure a product.

With feature models a predefined set of relations and dependencies between features can be expressed. Features can for example be mandatory, optional or alternative (see [3] for more details). Figure 1 illustrates a small example of how a feature model for a simple car could look like. The car has two mandatory features (indicated by the filled circle) named *Car body* and *Engine* and one optional feature (indicated by the empty circle) named *Multimedia system.*

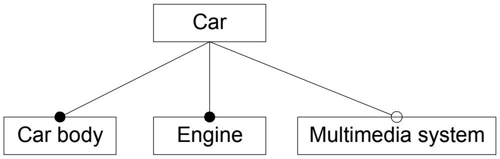

Figure 1: Feature model for a simple car

We argue that current feature-oriented modeling approaches have important limitations with respect to expressing the interactions of features. For expressing advanced kinds of feature interactions, the available set of feature relations within feature models is not sufficient. To illustrate our point we will give an example of a multimedia system for a car. The system consists of four optional features, named *Email system, DVD player, Radio,* and *Personalization* (as illustrated in Figure 2).

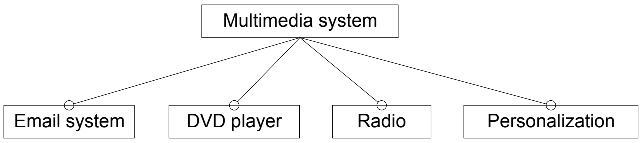

Figure 2: Feature model for a simple multimedia system

The feature *Personalization* allows different users of the multimedia system to configure it for their special needs and to store their preferences within the system. Personalization influences the feature Radio as a special (probably more expensive) type of radio has to be selected when Personalization is present. It allows the user to select a preferred radio station that can be stored within the radio. The feature Personalization influences the features DVD player and Email system in the same way as the selected type of player or email client has to support the storage of different user preferences. So the Personalization feature interacts with all other features in order to get information about the user preferences specific to each of these features. Within the solution domain (implementation model[1]), the implementation of the feature Personalization will crosscut the implementations of several other features. Personalization will therefore be called a *crosscutting concern* in the AO sense. As these crosscutting feature interactions can already be identified within the problem domain, we call the feature Personalization a *crosscutting feature.*

As stated before, current feature-oriented modeling approaches are weak with regard to expressing crosscutting feature interactions within the problem domain of software product lines. We argue that (crosscutting) feature interactions should be expressed in feature models within the problem domain. To capture interaction points of features we introduce the notion of *feature join points*. In order to provide these kinds of join points we offer mechanisms to describe the dynamic characteristics of features. A behavioral view of the system has to be offered in order to express the interactions among participating features. Important questions that arise in this context include: What is the right granularity level of feature join points? Does a textual description of feature join points (using e.g. XML) suffice?

---

[1] We use the terms solution domain and implementation model synonymously.

To manage variabilities in design and code, features and their relations or interactions have to be handled consistently within the problem domain and the solution domain of software product lines. Within the solution domain generative approaches can be used to build generative models for families of systems and generate concrete systems from these models. Our work focuses on two very powerful implementation technologies for generative programming, named aspect-oriented approaches (e.g. [4]) and generators (e.g. [5, 6]). Aspect-oriented approaches offer the possibility of not having to map certain features from the problem domain to sets of scattered little components in the solution domain. Interactions can be expressed using AOPs notion of join point interception. Crosscutting features can be mapped directly to single, well localized aspects in the solution domain. Generators, the second implementation technology we focus on, are programs that take a higher-level specification of a piece of software and produce its implementation [3]. GenVoca generators [6], for example, synthesize programs by composing modules that implement features. Distinct programs in the product line are described by distinct combinations of features.

Our work consists of three parts (as illustrated in Figure 3). The first part concentrates on specifying crosscutting feature interactions within the problem domain. So called *feature join points* should allow expressing complex feature interactions in feature models. For this purpose the dynamic characteristics of features have to be expressed explicitly. The second part concentrates on the design of code base assets for a software product line and their implementation model. It includes the mapping of (crosscutting) features to concrete software architectural building blocks (components, aspects, templates…). Feature join points are mapped to concrete join points in the AO sense. Within the last part support is provided to automatically generate concrete products using generative approaches for linking the software architectural building blocks. We want to investigate how far we get with generative approaches[2] and focus on answering the following question: Is it possible to automatically produce a concrete member of the product line without manually changing or writing any glue code?

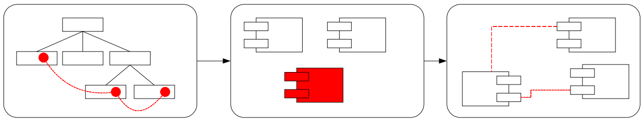

Figure 3: Product line development

The basic idea is to improve product line development and variability management using generative approaches to handle (crosscutting) feature interactions within each phase of the development process. Generative approaches allow the automatic generation of concrete products when used for component binding. The main goal of our work is to prove the assumption that generative approaches improve product line development and that they do a better job in capturing and composing individual features.

REFERENCES

[1] P. Clements and L. Northrop. Software Product Lines: practices and patterns. Addison-Wesley, 2002.

[2] K. Kang at al. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report, CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990.

[3] K. Czarnecki and U. Eisenecker. Generative programming: methods, tools, and applications. Addison-Wesley, 2000.

[4] M. Mezini and K. Ostermann. Variability Management with Feature-Oriented Programming and Aspects. Foundations of Software Engineering (FSE-12), ACM SIGSOFT, 2004.

[5] OpenArchitecturWare Generator Framework, http://sourceforge.net/projects/architecturware

[6] D. Batory and S. O'Malley. The design and implementation of hierarchical software systems with reusable components. ACM Transactions on Software Engineering and Methodology, 1(4):355-398, 1992.

---

[2] Within the domain of generative software development, we focus on aspect-oriented approaches and generators.