

Requirements, Features and Aspects for Software Product Lines

Klaus Alfert
Zühlke Engineering GmbH
Eschborn, Germany
klaus.alfert@zuehlke.com

Abstract

In this paper we report our experience with modeling requirements in software product lines. Variability of requirements are handled with feature models.

Non-functional requirements may have aspect-like characteristics. We show how they can be represented within feature models. A main result is that feature models provide low-level support for aspects and we discuss how they can be extended to provide better support for feature models with aspect characteristics.

1 Introduction

Software product lines [CN02, Bos00] focus on developing a family of related products by applying a systematic and managed re-use of development artifacts, often called *core asset base*, across the entire product family. Since the products vary, a major effort in developing software product lines is to manage the

variability and commonality of artifacts to enable the intended re-use. Variability occurs throughout the entire lifecycle and generally affects all re-usable artifacts, ranging from business cases, project plans and requirements to program code and test cases.

Aspects and aspect-oriented development (AOD) investigate with new modularization techniques, to separate concerns which are spread throughout an entire software system. These concerns are often entangled and intertwined if a system is structured through traditional development abstractions such as functions or classes. Aspects originate in the programming field but are currently moving towards other software engineering disciplines. The two article collections [FECA05, CRS⁺05] give a good overview of current results.

Aspects are a promising approach for software product lines in particular, as they can be used to improve modularization of variable and common structures in the core asset base and the products. In the following, we present our experience

with managing variability on the requirements level and discuss how this relates to aspect-oriented approaches.

2 Requirements

In the projects considered for this paper, we have used product lines for embedded systems such as medical devices or automotive systems to systematically exploit advantages in the development of closely related products. For medical devices in particular, a regulated development environment is required, which focuses on highly consistent documentation of development artifacts and their relationships (such as traces from requirements to implementation).

Embedded systems require resource optimizations such as cpu speed, memory footprint, or power consumption. This results in prominent non-functional requirements for both core assets and products. In traditional requirement management processes, e.g. based on Use Cases such as in the RUP [Kru98], non-functional requirements find their proper place in a special section of the requirements documents (e.g. the Supplementary Specification in the RUP).

However, when specifying or designing the system in more detail and more operationally, non-functional requirements have an impact on various areas of the system and their implementation is scattered throughout the entire system. In this situation non-functional requirements reveal their cross-cutting nature [CRS⁺05].

A similar effect occurs with the introduction of new technology concepts. Such concepts might be introduced due to updated legislation or because of new mechanical constraints due to a new

shape of the device under consideration to support the new marketing campaign. Often these new concepts occur as a maintenance step of a system. Adding such concepts to existing systems often results in modifications at several functional areas within these systems. In addition, they usually interact with the classical non-functional requirements mentioned above. Depending on the priority of and interaction with the non-functional requirements, the functional requirements of such new concepts may be different.

For single system development the complexity and interaction of functional and non-functional requirements does not pose a big problem, but things get complicated when they happen in a product line where we want to maintain a core asset base for all products. Variability of products includes often different constraints concerning resource usage or, as in our example, the product's shape, such that we required to provide these variants of functional requirements for the same concepts. This necessitates managing requirements very systematically.

3 Features and Variability

A systematical approach towards managing variability is to apply FODA-like feature modeling as discussed in [CE00]. Feature models employ a hierarchical structure of features as a refinement hierarchy with explicit notions of variability. This hierarchy is sometimes called a feature tree.

A feature f can be marked as an optional feature or as an alternative to another feature or to a set of other features, which have to be siblings of f . If a feature is optional or an alternative, this fea-

3 Features and Variability

ture is a variable feature, otherwise it is called a mandatory feature. In the hierarchy, these variability indications have only a local scope within the current subtree. To broaden this scope, two further relationships between features independent of their position in the hierarchy are introduced: *requires* and *excludes*. Applying these relationships results in a graph structure of features.

Concerning variability information, feature models can be transformed such that only *requires* and *excludes* vertices are used and features do not have to be marked as variable features. This transformed or canonicalized model is the basis for verifying the consistency of a set of selected features, e.g. the features of a particular product.

Feature models are a general approach to organize a set of notions and thus resemble object-oriented class models. They have proven to be helpful in our projects as model for requirements engineering within the context of software product lines. Here, each requirement is identified as a feature. The tree structure is a refinement relationship between features (i.e. requirements). Requirements that are only used for some products are marked as variable features. With *requires* and *excludes* we state additional constraints between variable features. This structure gives us the possibility to organize the requirements not only of a single product but of the entire product line within one model. A specific product is represented by a consistent selection of features, i.e. the product's requirements. Here, consistency is related only to the relationships between features.

The dependency between non-functional requirements and their operational counterparts can be modeled explicitly via feature relations,

since the non-functional requirements point via the *requires*-relation to their corresponding functional requirements.

	n_1	n_2	n_3
f_1		x	
f_2	x		x
g_2		x	x
g_2	x		

Table 1: *requires*-constraints within the feature model

As an example, consider the feature model given in Fig. 1, where we have functional and non-functional requirements. In this model, all leaf nodes are alternatives to their siblings. The *requires*-constraints between non-functional and functional requirements are shown in Table 1. The non-functional requirement n is refined into three alternatives n_1 , n_2 and n_3 . On the functional side, we have two requirements f and g which are refined into two alternatives f_1, f_2 and g_1, g_2 respectively. The *requires*-constraints show that each n_i identifies a set A_i of functional requirements, which realize n_i :

- $A_1 = \{f_2, g_2\}$
- $A_2 = \{f_1, g_1\}$
- $A_3 = \{f_2, g_1\}$

The linked non-functional and functional requirements share some important characteristics. In particular if a functional requirement is a variability, then all linked non-functional requirements have to be variable as well. This is required to be able to select all dependent functional requirements if the respective non-functional requirements are selected. In our example, this means that selecting

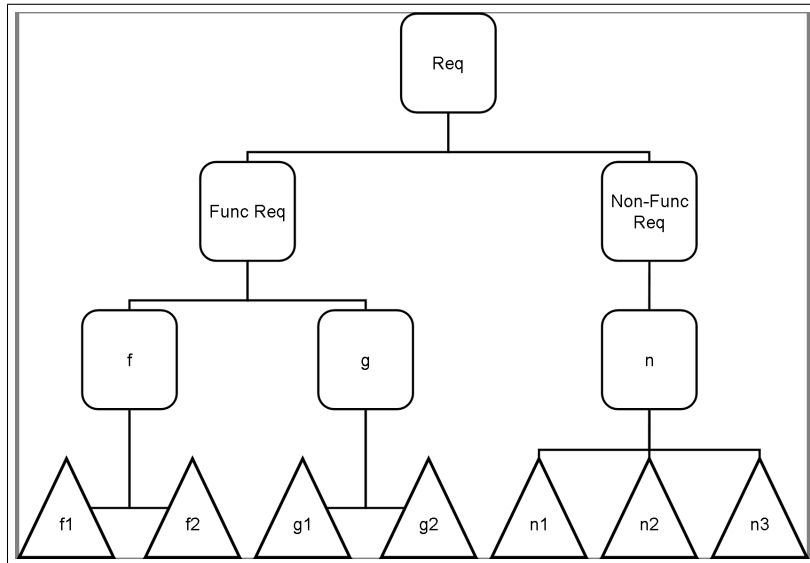


Figure 1: A sample feature model. Rounded rectangles are mandatory features, triangles denote alternative features.

n_2 results in selecting all features in A_2 as well.

4 Features and Aspects

Generally, aspect-oriented approaches fall roughly into two categories: describing aspects and composing aspects. The approaches focusing in programming, such as Aspect/J [KHH⁺01], are examples of the latter category, where we have means for both describing and composing aspects (the so called weaving) into an (existing) system. Modeling approaches, such as COSMOS [SR05], are instances of the former category. They do not focus on how the separated concerns are systematically composed into a system.

Feature models, as discussed above, give us the possibility to model functional and non-functional requirements,

their variability and their interdependency for an entire product line. The cross-cutting nature of non-functional requirements is immediately accessible by the *requires*-relations to their functional counterparts. Due to this dependency, a selection of non-functional requirement n_i results immediately in the selection of the set A_i of functional requirements, modifying—if selected—the product’s specification in several well-defined places. Thus, the pairs (n_i, A_i) resemble aspects.

However, feature models, as we have used them, provide only a somewhat immature support for aspects. This is not surprising, since feature models have not been developed with aspects in mind. Concern modeling is possible with feature models, however, it is limited to the mapping from non-functional to functional requirements. The non-functional

References

requirements serve as concerns, which influence dependent functional requirements. But the identification of non-functional requirements as a concern or an aspect is up to the analyst, since feature models neither provide concerns nor aspects as first-class model entities and, of course, not every non-functional requirement qualifies as a concern.

Feature models provide a limited composition operator. Combining aspects required for a particular product results in applying an advice which can roughly be described as *select this set of features instead of that set*. Due to the transitivity of the *requires*-relation, a cascade of further feature selection may happen. The limited characteristic of this advice is that the connection between what features are selected and where they have to be selected cannot be defined outside the feature model, but is an intrinsic model information. Additionally, there are no means to prescribe how the composition shall work, such as adding additional constraints or providing new variable features.

5 Summary and Future Investigations

FODA-based feature models provide a practically usable model for variability of requirements within an entire software product line. They even provide some support for aspect characteristics of requirements, but only on a low level of abstraction.

In our experience, this low level of abstraction cannot be raised without substantially modifying the FODA-based approach. The reason for this is a general weakness in the feature model: The re-

finement relationship between nodes in a feature model has only a weak semantic basis. There is no differentiation between *consists of* and *specializes* relations as for example used in object-oriented models. Similarly, there are no rules how a refinement or variation is related to parent nodes, such as the design by contract rules define for proper object-oriented refinements [Mey88]. Thus, it depends only on the modeler's discipline how the feature model is structured. As a consequence, there is no possibility to specify generically how the feature model shall be modified by applying an aspect without explicit knowledge of how the feature model looks like.

With such modifications of the feature model, aspects, concerns, and compositional operators can be introduced as first-class entities of feature models. This would be an important step towards systematic re-use of aspects, and of using aspects as granule of re-use in software product lines.

References

- [Bos00] Bosch, Jan: *Design & Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. Addison-Wesley, Harlow, 2000.
- [CE00] Czarnecki, Krzysztof and Ulrich W. Eisenecker: *Generative Programming: Methods, Tools and Applications*. Addison-Wesley, 2000.
- [CN02] Clements, Paul and Linda Northrop: *Software Product Lines: Practices and Patterns*. SEI Series in Software-Engineering.

References

- Addison-Wesley, Boston, 2002.
- [CRS⁺05] Chizchyan, Ruzanna, Awais Rashid, Pete Sawyer, Alessandro Garcia, Mónica Pinto Alarcon, Jethro Bakker, Bedir Tekinerdoğan, Siobhán Clarke, and Andrew Jackson: *Survey of aspect-oriented analysis and design approaches*. Technical Report AOSD-Europe-ULANC-9, AOSD-Europe: EU Network of Excellence, May 2005.
- [FECA05] Filman, Robert E., Tzilla Elrad, Siobhán Clarke, and Mehmed Aksit (editors): *Aspect-Oriented Software Development*. Addison-Wesley, Boston, 2005.
- [KHH⁺01] Kizcales, Gregor, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Grishold: *Getting started with aspect/j*. Communications of the ACM, 44(10), October 2001.
- [Kru98] Kruchten, Philippe: *The Rational Unified Process: An Introduction*. Addison-Wesley, 1998.
- [Mey88] Meyer, Bertrand: *Object-Oriented Software Construction*. Prentice-Hall, 1988.
- [SR05] Sutton Jr., Stanley M. and Isabelle Rouvellou: *Concern modeling for aspect-oriented software development*. In Filman, Robert E. *et al.* [FECA05], chapter 21, pages 479–505.